NAME

JavaBridge − Enable invocation of java classes from C

SYNOPSIS

Enable C to talk with the protocol exposed by php-java-bridge. Alternatly, allow C to call java without complex JNI.

DESCRIPTION

The *javabridge* library is a C binding of the communication protocol utilised by *php-java-bridge.*

(Please refer to *http://php-java-bridge.sf.net* for that project)

The motivation for this library was frustration with other methods of getting C to invoke java. I find JNI with its dependancy on java versions and etc. to be cumbersome - particularly when cross compiling. Also with either JNI or a direct system call to java there is an overhead for each invocation with the JVM instantiating. To do RPC / SOAP calls to a java application server requires Tomcat or the like running, and for any simple application I again found this to be cumbersome.

The good people at php-java-bridge.sf.net have built an excellent pure java application that can either be installed into a java application server or can be run in a standalone fashion. Their project provides a bridge allowing PHP to call java. Their method is a simple XML communication from PHP to the java bridge application. The application takes care of reflecting java classes called for, and sends the answers back to PHP.

This library is an C implimentation of that communication protocol. Additionally, this library can fork the java bridge application in standalone mode so that a java servlet engine is not required.

In the event you wish to run java-pridge in a proper application server like Tomcat, please refer to the extensive documentation available at *http://php-java-bridge.sf.net* The documentation herein is confined to starting a stand-alone application server, and the bindings provided to enable C to talk to that server in-order to invoke java from C.

LIBRARY DETAIL

Environment variables

JAVABRIDGE_LISTENPORT overrides the compiled in default TCP port (9267) that the java application server is (or should) listening on.

JAVABRIDGE_CONFPATH overrides the location of javabridge.conf. This should be an absolute path to the relevant file. If this exists it overrides the path passed to the library from *startJservice*

CLASSPATH makes additional java classes available to the java application server. Normal java syntax for CLASSPATH on your platform applies

TypeDefs in the library

javaConnection is an int. Carries an opened socket to the java application server.

javaObject is an unsigned int. Carries an particular java class or object.

Defined return values

jERROR === 0 *jFALSE* === -1 *jTRUE* === -2 *jVOID* === -3

Java application server functions

javaConnection startJservice (char *confPath)

This should be the first call to the libarary.

It will first test if the java application server is listening on the configured port, and if not it will start the server. It then opens communication to the running java application server

The function will abort (i.e. exit with failure) under the following circumstances.

JavaBridge.jar is not locatable on the classpath definined in either javabridge.conf or by the CLASSPATH environment variable.

On windows, TCP/IP initialisation failure occurs.

Inability to create a socket.

Communication is established to some service at JAVABRIDGE_LISTENPORT, but that service does not respond as expected.

When no servce is found at JAVABRIDGE_LISTENPORT, and a fork of the standalone java application server fails.

A standalone java application server is started but does not respond as expected within 120 seconds.

It returns a handle to talk to the service with. In reality the handle is a connected socket.

void stopJservice (javaConnection)

This should be the last call to the libarary.

It closes communication with the java application server. The paramter is the socket that is returned when opening a connection to the server.

Note: this DOES NOT stop the java application server running. If you REALLY want to stop the application server, see *stopJservice* below. Obviously, after this nothing will work, including *stopJservice*

void killJservice (javaConnection)

This shouldn't be called unless you REALLY want to stop the JavaBridge service.

It forces a halt of the JVM and any stuff opened by the JVM will be left where they were. The paramter is the socket that is returned when opening a connection to the server.

Note: this STOPS the java application server uncleanly. I don't recommend that you do this to an instance running in Tomcat or the like, however it will successfully kill a standalone invocation. Obviously, after this nothing will work, including *stopJservice*

Java bridge communication functions

javaObject getJclass (javaConnection sock, javaObject *jException, char *jClass)

Allows manipulation of a java class without instatitating an object. Java experts can help with when to do this. A good example is when you want to do java.lang.Class.forName("someClass").

Input is a handle of an open socket to the server, the address of an exception javaObject, and a string defining the class desired, for example "java.sql.DriverManager".

It returns type javaObject, or any of the jXXXX answers. If jERROR (== 0) is returned then jException will refer to a java exception that can be interrogated for more detail.

Note: The return is NOT a real java object, and is only used as a refererence to static class members.

javaObject constructJobject (javaConnection sock, javaObject *jException, char *jClass, char *formatStr, ...)

Creates an instance (or object) from a class. So:

```
javaObject aString = constructJobject(jvm, &anException,
      "java.lang.String", "%s", "Hello World");
```

is equivalant to the java statement

```
aString = new String("Hello World");
```

Input is a handle of an open socket to the server, the address of an exception javaObject, a string defining the class desired, a format string defining paramaters to the java class instantition, and the actual paramaters to the java class. (See *Parameters&formats* below for more information about this).

It returns type javaObject, or 0 on failure. If jERROR (== 0) is returned then jException will refer to a java exception that can be interrogated for more detail.

javaObject invokeJobject (javaConnection sock, javaObject *jException, javaObject jObj, char *jMethod, char *formatStr, ...)

Allows calling a method in an instantiated java class (or object), or calling a static method in a java reference class. So:

```
javaObject aNum_a = constructJobject(jvm, &anException,
      "java.math.BigInteger", "%s", "6");
javaObject aNum_b = constructJobject(jvm, &anException,
      "java.math.BigInteger", "%d", 1);
```

javaObject aNum_c invokeJobject(jvm, &anException,
aNum_a, add , %o , aNum_b);

is equivalant to the java

```
import java.math.*;
aNum_a = new BigInteger("6");
aNum_b = new BigInteger(1);
```

aNum_c = aNum_a.add(aNum_b);

Input is a handle of an open socket to the server, the address of an exception javaObject, a previously instantiated object, a string defining the object method desired, a format string defining paramaters to the java method, and the actual paramaters for the method. (See *Parameters&formats* below for more information about this).

It returns type javaObject, or any of the jXXXX answers. If jERROR (== 0) is returned then jException will refer to a java exception that can be interrogated for more detail.

int setJproperty (javaConnection sock, javaObject jObj, char *jProperty, char *formatStr, ...)

Allows setting an object property. I haven't found use for this yet, but php-java-bridge impliments it, and I guess a java expert will know when to do this.

Input is a handle of an open socket to the server, the address of an exception javaObject, a previously instantiated object, a string defining the object property desired, a format string defining paramaters to the java property, and the actual paramaters for the property. (See *Parameters&formats* below for more information about this). I guess by definition you should only pass one parameter to a class property.

It returns non zero on success, or jERROR on failure. If jERROR (= 0) is returned then jException will refer to a java exception that can be interrogated for more detail.

void releaseJobject (javaConnection sock, javaObject jObj)

Tells the java application server that you are done with this construct. Allows garbage collection in the invoked JVM for this construct. In short, if you create it, then release it.

Input is a handle of an open socket to the server, and previously instantiated object

char *getJexception (javaConnection sock, javaObject jObj)

Given that some funtion has returned jERROR (== 0), return a string representation of the java exception.

> Note: you should free() this string when you are done with it.
>
> Note: you should also releaseJobject on the jException object when you are done with it to allow the JVM to do garbage collection.

Input is a handle of an open socket to the server, and a previously set jException object.

It returns a char pointer to the error string on success, or NULL otherwise.

void releaseJobject (javaConnection sock, javaObject jObj)

Tells the java application server that you are done with this construct. Allows garbage collection in the invoked JVM for this construct. In short, if you create it, then release it. Remeber to invoke close() or release or whatever java expects of the object to clear references in the JVM to this object first.

Input is a handle of an open socket to the server, and previously instantiated object.

char *getJexception (javaConnection sock, javaObject jObj)

Given that some funtion has returned jERROR (== 0), return a string representation of the java exception.

> Note: you should free() this string when you are done with it.
>
> Note: you should also releaseJobject on the jException object when you are done with it to allow the JVM to do garbage collection.

int getJboolean (javaConnection sock, javaObject jObj)

Attempt to get a BOOLEAN (TRUE == !0 or FALSE == 0) result from the given object.

Input is a handle of an open socket to the server, and previously instantiated object.

Will return either a 0 or a 1.

char *getJstring (javaConnection sock, javaObject jObj)

Attempt to get a string (char *) result from the given object.

> Note: you should free() this string when you are done with it.

Input is a handle of an open socket to the server, and previously instantiated object.

Will return either a 0 or a 1.

int64_t getJlong (javaConnection sock, javaObject jObj)

Attempt to get a long (int64_t) result from the given object.

Input is a handle of an open socket to the server, and previously instantiated object.

Will return (as best as it can) a (signed) numeric representation of the requested class.

double getJdouble (javaConnection sock, javaObject jObj) Attempt to get a floating (double) result from the given object.

Input is a handle of an open socket to the server, and previously instantiated object.

Will return (as best as it can) a double representation of the requested class. On failure returns C "nan".

Parameters&formats

Instatiation of java classes or calls to java methods in a object / class often need parameters. This library handles that requirement similarly to printf() and friends. The first parameter is a string with format flags defining how the remaining parameters should be treated.

Anything after the (required) format string is interpreted as a particular C type and passed through to the java application server as a representation of that parameter.

Where a parameter can be expexted but you don’t wish or need to send one, pass a NULL to the format string parameter.

Format strings currently understood are:

> %o - send a "java object" or "class" to the JVM

%s - send a string to the JVM

%d - send an integer to the JVM

%f - send a double (or float) to the JVM

%b - send a boolean (0 or non zero) to the JVM

EXAMPLES

Config file

```
# javabridge.conf
# Both # and ; are treated as start of comment flags in this file
# You MUST have at least JavaBridge.jar in the class path definition
# You can do this by setting "CLASSPATH" environment variable, or as
#  below with the "classpath" configure entries or by dropping it into
#  the basejarsearch directory defined below.
#  (this last is probably the easiest)
# If you want JDBC (or any other class external to your JVM) you must
#   include the relevant jar on "CLASSPATH" environment variable, or
#   in this file as below.
# Note: that environment "CLASSPATH" is included AFTER any entries below.

# The location of this file can be defined by setting "JAVABRIDGE_CONFPATH"
#  environment variable.

# If you want JDBC you should use Class.forName(my.company.driver)
#   to load it into the JVM's DriverManager from the class path at
#   runtime.

basejarsearch=C:/msysOptMount/winExtra/lib/javabridge/jars

#classpath=C:/msysOptMount/winExtra/lib/javabridge/jars/JDBC/sqliteJDBC/sqlite-jdbc-
#classpath=/SomeDir/JDBC/UDB2JDBC/db2jcc4.jar
```

Code

```
// testBridge.c
// The best domentation is an example

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>

#include "javabridge.h"

int
main(int argc, char **argv)
{

      char myRoot[PATH_MAX];
      char confFile[PATH_MAX];
```

```
// Stand alone application - find out where the application is
//  running from so that we can pass an absolute path to THIS apps
//   configuration instead of using  the default
//   /usr/local/etc/javabridge/ configuration.
// Note: if you installed the library into a non-standard
//  location (e.g. /opt/test/) and you want a site wide
//  configuration you sould define or set confFile to
// "/opt/test/etc/javabridge/javabridge.conf"

struct stat aFileD;

#if defined(WIN32) || defined(_WIN32)
      #define PATHSEPERATOR ";"
#else
      #define PATHSEPERATOR ":"
#endif  // WIN32

strcpy(confFile, argv[0]);

#if defined(WIN32) || defined(_WIN32)
      // Windows nonsense
      int i;
      for (i=0; i < strlen(confFile); i++)
            if (confFile[i] == '\')
                  confFile[i] = '/';
#endif
if (confFile[0] == '/' || (confFile[1] == ':' && confFile[2] == '/'))
      // called with an absolute invocation
      strcpy(myRoot, confFile);
else
{
      // called with an relative path invocation
      if (strchr(confFile, '/'))
      {
            // called with an relative path invocation
            getcwd(myRoot, sizeof(myRoot));
            strcat(myRoot, "/");
            strcat(myRoot, confFile);
      }
      else
      {
            //check current dir (may not have . on the path)
            getcwd(myRoot, sizeof(myRoot));
            strcat(myRoot, "/");
            strcat(myRoot, confFile);
            if(stat(myRoot, &aFileD) == -1)
            {
                  // called without path, search path to find ourselves
                  char *aPath;
                  char *sysPath = getenv("PATH");
                  if (sysPath != NULL)
                  {
                        aPath = strtok(sysPath, PATHSEPERATOR);
                        while (aPath != NULL)
```

```
                        {
                              strcpy(myRoot, aPath);
                              strcat(myRoot, "/");
                              strcat(myRoot, confFile);
                              if(stat(myRoot, &aFileD) != -1)
                                    break;
                              aPath = strtok(NULL, PATHSEPERATOR);
                        }
                        free(sysPath);
                  }
            }
      }
}

*(strrchr(myRoot, '/')) = 0;
strcpy(confFile, myRoot);
strcat(confFile, "/javabridge.conf");

// Information about which .conf file is used
fprintf(stderr, "ConfFile:%s0, confFile);
fflush(stderr);

// We already have something listening on the default port of 9267,
//  override it here to an unused port
//  (OK not really - but for examples sake)
putenv("JAVABRIDGE_LISTENPORT=9267"); //no setenv in MinGW!

// Start the php-java-bridge listener
// If its already running, use it otherwise fork a call to start it.

// Three ways to set conf file location to the library:
#if defined(USEJENVIRONMENT)
      // Via environment variable
      strcpy(confFile, "JAVABRIDGE_CONFPATH=");
      strcat(confFile, myRoot);
      putenv(confPath);
      javaConnection jvm = startJservice(NULL);
#else
      // Direct invocation
      javaConnection jvm = startJservice(confFile);
#endif
// Or export JAVABRIDGE_CONFPATH in a wrapping shell script
//  and skip all of the above path nonsense


// Should now have the JavaBridge listening on 9267
// Do some stuff with it.

// Declare something to catch errors with
//  Note this merely an unsigned int, but JavaBridge.jar keeps
//   objects as such, and it writes any exception as an instance of
//   exception, so to work with errors we need to remember what it.
javaObject anException;
```

```
// Simple test to see if we have something working here

// Construct a Long object.
// Note: the parameters to the constructor which creates the object
//   with value 6
javaObject myLong = constructJobject(jvm, &anException,
      "java.lang.Long", "%d", 6);
// Report errors. Should probably exit(FAIL) here cause nothing
//   further will work if we cannot get past here.
// Note: getException allocates space for the returned string,
//   remember to free it.
if (!myLong)
{ char *e = getJexception(jvm, anException);
      printf("Failed: %s0, e);      free(e);
}
// Get the value of the object.
char *ans = getJstring(jvm, myLong);
if (ans && !strcmp(ans, "6"))
      printf("PASS (getJstring) *** Got:%s Expected '6'0, ans);
else
      printf("FAIL (getJstring) *** Got:%s Expected '6'0, ans);
// getString dynamically allocates space for the returned
//   answer, remember to free it.
free(ans);
// Test some other mappings for java values to C values in
//   the library
if (getJlong(jvm, myLong) == 6)
      printf("PASS (getJlong) *** Got:%lld Expected '6'0,
        getJlong(jvm, myLong));
else
      printf("FAIL (getJlong) *** Got:%lld Expected '6'0,
        getJlong(jvm, myLong));
if (getJdouble(jvm, myLong) == 6.0)
      printf("PASS (getJdouble) *** Got:%f Expected '6.0'0,
        getJdouble(jvm, myLong));
else
      printf("FAIL (getJdouble) *** Got:%f Expected '6.0'0,
        getJdouble(jvm, myLong));
// Release what we have used so that the JVM can do garbage
//    collection.
releaseJobject(jvm, myLong);


// More complex example - do some JDBC stuff.

// Get the class definition for "Class"
//   Note: this is not creating an
//     object (java.lang.Class cannot be instantiated)
javaObject aClass = getJclass(jvm, &anException, "java.lang.Class");
if (!aClass)
      { char *e = getJexception(jvm, anException);
            printf("Failed: %s0, e);
            free(e);
       }
```

```
else
      printf("PASS (getJclass) *** Got:java.lang.Class0);
// Dynamically load a JDBC driver.
//   Note: this is still not creating a java object, simply calling
//   the static method "forName" of the class "Class"
javaObject jdbcDrv = invokeJobject(jvm, &anException, aClass,
  "forName", "%s", "org.sqlite.JDBC");
if (!jdbcDrv)
      { char *e = getJexception(jvm, anException);
            printf("Failed: %s0, e);
            free(e);
      }
else
      printf("PASS (invokeJobject) *** Got:java.lang.Class.forName(org.sqlite.
// We have loaded the driver, get rid of what we used to do so.
releaseJobject(jvm, jdbcDrv);
releaseJobject(jvm, aClass);

// Find the DriverManager class
javaObject manager = getJclass(jvm, &anException,
  "java.sql.DriverManager");
if (!manager)
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (getJclass) *** Got:java.sql.DriverManager0);
// Use DriverManager.getConnection to instantiate our first
//   real java object - a JDBC connection.
javaObject connection = invokeJobject(jvm, &anException, manager,
  "getConnection", "%s", "jdbc:sqlite:testJDBC.sdb");
if (!connection)
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:java.sql.DriverManager.getConnectio
// Do some stuff with the connection object
invokeJobject(jvm, &anException, connection, "setAutoCommit",
  "%b", 1);
// and get a new object from the connection object so that we can
//  do useful stuff
javaObject statement = invokeJobject(jvm, &anException, connection,
  "createStatement", NULL);
if (!statement)
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:Connection.createStatement()0);
// Do some useful stuff
```

```
invokeJobject(jvm, &anException, statement, "execute", "%s",
  "drop table test1");
if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
   "create table test1(col1 char(10), col2 decimal(11.2) )"))
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:Statement.execute(create table)0);
if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
   "insert into test1 values('rowThe1st', 99.99)"))
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:Statement.execute(insert)0);
if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
  "insert into test1 values('rowThe2nd', 0.01)"))
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:Statement.execute(insert)0);
// OK, now try and get something out of the data we have written
if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
    "select sum(col2) as tot from test1"))
{ char *e = getJexception(jvm, anException);
      printf("Failed: %s0, e);
      free(e);
}
javaObject resultset = invokeJobject(jvm, &anException, statement,
    "getResultSet", NULL);
if (!statement)
{ char *e = getJexception(jvm, anException);
      printf("Failed: %s0, e);
      free(e);
}
invokeJobject(jvm, &anException, resultset, "next", NULL);
javaObject aString = invokeJobject(jvm, &anException, resultset,
  "getString", "%s", "tot");
if (getJdouble(jvm, aString) == 100.0)
      printf("PASS (jdbc)*** Got:%s Expected 100.000,
        getJstring(jvm, aString));
else
      printf("FAIL (jdbc)*** Got:%s Expected 100.000,
        getJstring(jvm, aString));
// Release what we have used so the JVM can do garbage collection
releaseJobject(jvm, aString);
// Close our result set and allow the JVM to garbage collect it
invokeJobject(jvm, &anException, resultset, "close", NULL);
releaseJobject(jvm, resultset);
```

```
        // Test that we are in fact getting exceptions correctly.
        if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
          "create table test1(col1 char(10), col2 decimal(11.2) )"))
        {
              char *e = getJexception(jvm, anException);
              printf("PASS (exception)*** Got:%s 0, e);
              printf("  Expected ’SQLException: table test1 already exists’0);
              free(e);
        }
        else
        {
              printf("FAIL (exception)*** Got:Success ");
              printf("Expected ’SQLException: table test1 already exists’0);
        }
        // Clean up
        invokeJobject(jvm, &anException, statement, "close", NULL);
        releaseJobject(jvm, statement);
        invokeJobject(jvm, &anException, connection, "close", NULL);
        releaseJobject(jvm, connection);
        releaseJobject(jvm, manager);


        // And drop our connection to the running application server
        //   Note this doesn’t really stop the server running, it simply
        //    tells the server that we are not interested in further
        //    communication, and that it can clean up, close sockets and etc.
        stopJservice(jvm);

        // OR if you really don’t need the service anymore do an ugly thing
        // and kill it dead.
        //   Note this really stops the server running in an unclean way,
        //    and anything opened by the server will be dropped where it is.

        // Cause I am testing & testing & testing, I don’t want the service
        //    left running.

        // Got to get a new connections, as it it is dropped above.
        jvm = startJservice(confFile);
        // And kill it
        killJservice(jvm);

}
```

SEE ALSO

http://php-java-bridge.sf.net

REPORTING PROBLEMS

I am a long way from being fluent in java. If your issue is with java consult the java api docs, or look at oracles web site.

There is *javabridgeservice.bat* (for windows) and *javabridgeservice.sh* (for unix like OS) in install_path/etc. These scripts allow interactivly starting JavaBridge.jar with debugging turned on from a console. You may get a better idea of where things are failing by inspecting the output here while running your code.

php-java-bridge has a mailing list for users and comprehensive documentation for their bridge on source forge.

If you are certain that you have found a bug in the C library, *dirk@ddtdebuggers.co.za*

AUTHORS

Version 0.9, 2015/05/05

This software interfaces with php-java-bridge provided at *http://php-java-bridge.sf.net* While this library is not afilliated to that group, it is useless without the package that they supply. To my best understanding *php-java-bridge* is licensed for all use without restriction.

My deepest appreciation goes to the php-java-bridge team.

NAME

JavaBridge − Enable invocation of java classes from C

SYNOPSIS

Enable C to talk with the protocol exposed by php-java-bridge. Alternatly, allow C to call java without complex JNI.

DESCRIPTION

The *javabridge* library is a C binding of the communication protocol utilised by *php-java-bridge.*

(Please refer to *http://php-java-bridge.sf.net* for that project)

The motivation for this library was frustration with other methods of getting C to invoke java. I find JNI with its dependancy on java versions and etc. to be cumbersome - particularly when cross compiling. Also with either JNI or a direct system call to java there is an overhead for each invocation with the JVM instantiating. To do RPC / SOAP calls to a java application server requires Tomcat or the like running, and for any simple application I again found this to be cumbersome.

The good people at php-java-bridge.sf.net have built an excellent pure java application that can either be installed into a java application server or can be run in a standalone fashion. Their project provides a bridge allowing PHP to call java. Their method is a simple XML communication from PHP to the java bridge application. The application takes care of reflecting java classes called for, and sends the answers back to PHP.

This library is an C implimentation of that communication protocol. Additionally, this library can fork the java bridge application in standalone mode so that a java servlet engine is not required.

In the event you wish to run java-pridge in a proper application server like Tomcat, please refer to the extensive documentation available at *http://php-java-bridge.sf.net* The documentation herein is confined to starting a stand-alone application server, and the bindings provided to enable C to talk to that server in-order to invoke java from C.

LIBRARY DETAIL

Environment variables

JAVABRIDGE_LISTENPORT overrides the compiled in default TCP port (9267) that the java application server is (or should) listening on.

JAVABRIDGE_CONFPATH overrides the location of javabridge.conf. This should be an absolute path to the relevant file. If this exists it overrides the path passed to the library from *startJservice*

CLASSPATH makes additional java classes available to the java application server. Normal java syntax for CLASSPATH on your platform applies

TypeDefs in the library

javaConnection is an int. Carries an opened socket to the java application server.

javaObject is an unsigned int. Carries an particular java class or object.

Defined return values

jERROR === 0 *jFALSE* === -1 *jTRUE* === -2 *jVOID* === -3

Java application server functions

javaConnection startJservice (char *confPath)

This should be the first call to the libarary.

It will first test if the java application server is listening on the configured port, and if not it will start the server. It then opens communication to the running java application server

The function will abort (i.e. exit with failure) under the following circumstances.

JavaBridge.jar is not locatable on the classpath definined in either javabridge.conf or by the CLASSPATH environment variable.

On windows, TCP/IP initialisation failure occurs.

Inability to create a socket.

Communication is established to some service at JAVABRIDGE_LISTENPORT, but that service does not respond as expected.

When no servce is found at JAVABRIDGE_LISTENPORT, and a fork of the standalone java application server fails.

A standalone java application server is started but does not respond as expected within 120 seconds.

It returns a handle to talk to the service with. In reality the handle is a connected socket.

void stopJservice (javaConnection)

This should be the last call to the libarary.

It closes communication with the java application server. The paramter is the socket that is returned when opening a connection to the server.

Note: this DOES NOT stop the java application server running. If you REALLY want to stop the application server, see *stopJservice* below. Obviously, after this nothing will work, including *stopJservice*

void killJservice (javaConnection)

This shouldn't be called unless you REALLY want to stop the JavaBridge service.

It forces a halt of the JVM and any stuff opened by the JVM will be left where they were. The paramter is the socket that is returned when opening a connection to the server.

Note: this STOPS the java application server uncleanly. I don't recommend that you do this to an instance running in Tomcat or the like, however it will successfully kill a standalone invocation. Obviously, after this nothing will work, including *stopJservice*

Java bridge communication functions

javaObject getJclass (javaConnection sock, javaObject *jException, char *jClass)

Allows manipulation of a java class without instatitating an object. Java experts can help with when to do this. A good example is when you want to do java.lang.Class.forName("someClass").

Input is a handle of an open socket to the server, the address of an exception javaObject, and a string defining the class desired, for example "java.sql.DriverManager".

It returns type javaObject, or any of the jXXXX answers. If jERROR (== 0) is returned then jException will refer to a java exception that can be interrogated for more detail.

Note: The return is NOT a real java object, and is only used as a refererence to static class members.

javaObject constructJobject (javaConnection sock, javaObject *jException, char *jClass, char *formatStr, ...)

Creates an instance (or object) from a class. So:

```
javaObject aString = constructJobject(jvm, &anException,
      "java.lang.String", "%s", "Hello World");
```

is equivalant to the java statement

```
aString = new String("Hello World");
```

Input is a handle of an open socket to the server, the address of an exception javaObject, a string defining the class desired, a format string defining paramaters to the java class instantition, and the actual paramaters to the java class. (See *Parameters&formats* below for more information about this).

It returns type javaObject, or 0 on failure. If jERROR (== 0) is returned then jException will refer to a java exception that can be interrogated for more detail.

javaObject invokeJobject (javaConnection sock, javaObject *jException, javaObject jObj, char *jMethod, char *formatStr, ...)

Allows calling a method in an instantiated java class (or object), or calling a static method in a java reference class. So:

```
javaObject aNum_a = constructJobject(jvm, &anException,
      "java.math.BigInteger", "%s", "6");
javaObject aNum_b = constructJobject(jvm, &anException,
      "java.math.BigInteger", "%d", 1);
```

javaObject aNum_c invokeJobject(jvm, &anException,
aNum_a, add , %o , aNum_b);

is equivalant to the java

```
import java.math.*;
aNum_a = new BigInteger("6");
aNum_b = new BigInteger(1);
```

aNum_c = aNum_a.add(aNum_b);

Input is a handle of an open socket to the server, the address of an exception javaObject, a previously instantiated object, a string defining the object method desired, a format string defining paramaters to the java method, and the actual paramaters for the method. (See *Parameters&formats* below for more information about this).

It returns type javaObject, or any of the jXXXX answers. If jERROR (== 0) is returned then jException will refer to a java exception that can be interrogated for more detail.

int setJproperty (javaConnection sock, javaObject jObj, char *jProperty, char *formatStr, ...)

Allows setting an object property. I haven’t found use for this yet, but php-java-bridge impliments it, and I guess a java expert will know when to do this.

Input is a handle of an open socket to the server, the address of an exception javaObject, a previously instantiated object, a string defining the object property desired, a format string defining paramaters to the java property, and the actual paramaters for the property. (See *Parameters&formats* below for more information about this). I guess by definition you should only pass one parameter to a class property.

It returns non zero on success, or jERROR on failure. If jERROR (= 0) is returned then jException will refer to a java exception that can be interrogated for more detail.

void releaseJobject (javaConnection sock, javaObject jObj)

Tells the java application server that you are done with this construct. Allows garbage collection in the invoked JVM for this construct. In short, if you create it, then release it.

Input is a handle of an open socket to the server, and previously instantiated object

char *getJexception (javaConnection sock, javaObject jObj)

Given that some funtion has returned jERROR (== 0), return a string representation of the java exception.

> Note: you should free() this string when you are done with it.
>
> Note: you should also releaseJobject on the jException object when you are done with it to allow the JVM to do garbage collection.

Input is a handle of an open socket to the server, and a previously set jException object.

It returns a char pointer to the error string on success, or NULL otherwise.

void releaseJobject (javaConnection sock, javaObject jObj)

Tells the java application server that you are done with this construct. Allows garbage collection in the invoked JVM for this construct. In short, if you create it, then release it. Remeber to invoke close() or release or whatever java expects of the object to clear references in the JVM to this object first.

Input is a handle of an open socket to the server, and previously instantiated object.

char *getJexception (javaConnection sock, javaObject jObj)

Given that some funtion has returned jERROR (== 0), return a string representation of the java exception.

> Note: you should free() this string when you are done with it.
>
> Note: you should also releaseJobject on the jException object when you are done with it to allow the JVM to do garbage collection.

int getJboolean (javaConnection sock, javaObject jObj)

Attempt to get a BOOLEAN (TRUE == !0 or FALSE == 0) result from the given object.

Input is a handle of an open socket to the server, and previously instantiated object.

Will return either a 0 or a 1.

char *getJstring (javaConnection sock, javaObject jObj)

Attempt to get a string (char *) result from the given object.

> Note: you should free() this string when you are done with it.

Input is a handle of an open socket to the server, and previously instantiated object.

Will return either a 0 or a 1.

int64_t getJlong (javaConnection sock, javaObject jObj)

Attempt to get a long (int64_t) result from the given object.

Input is a handle of an open socket to the server, and previously instantiated object.

Will return (as best as it can) a (signed) numeric representation of the requested class.

double getJdouble (javaConnection sock, javaObject jObj) Attempt to get a floating (double) result from the given object.

Input is a handle of an open socket to the server, and previously instantiated object.

Will return (as best as it can) a double representation of the requested class. On failure returns C "nan".

Parameters&formats

Instatiation of java classes or calls to java methods in a object / class often need parameters. This library handles that requirement similarly to printf() and friends. The first parameter is a string with format flags defining how the remaining parameters should be treated.

Anything after the (required) format string is interpreted as a particular C type and passed through to the java application server as a representation of that parameter.

Where a parameter can be expexted but you don’t wish or need to send one, pass a NULL to the format string parameter.

Format strings currently understood are:

> %o - send a "java object" or "class" to the JVM

%s - send a string to the JVM

%d - send an integer to the JVM

%f - send a double (or float) to the JVM

%b - send a boolean (0 or non zero) to the JVM

EXAMPLES

Config file

```
# javabridge.conf
# Both # and ; are treated as start of comment flags in this file
# You MUST have at least JavaBridge.jar in the class path definition
# You can do this by setting "CLASSPATH" environment variable, or as
#  below with the "classpath" configure entries or by dropping it into
#  the basejarsearch directory defined below.
#  (this last is probably the easiest)
# If you want JDBC (or any other class external to your JVM) you must
#   include the relevant jar on "CLASSPATH" environment variable, or
#   in this file as below.
# Note: that environment "CLASSPATH" is included AFTER any entries below.

# The location of this file can be defined by setting "JAVABRIDGE_CONFPATH"
#  environment variable.

# If you want JDBC you should use Class.forName(my.company.driver)
#   to load it into the JVM’s DriverManager from the class path at
#   runtime.

basejarsearch=C:/msysOptMount/winExtra/lib/javabridge/jars

#classpath=C:/msysOptMount/winExtra/lib/javabridge/jars/JDBC/sqliteJDBC/sqlite-jdbc-
#classpath=/SomeDir/JDBC/UDB2JDBC/db2jcc4.jar
```

Code

```
// testBridge.c
// The best domentation is an example

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>

#include "javabridge.h"

int
main(int argc, char **argv)
{

      char myRoot[PATH_MAX];
      char confFile[PATH_MAX];
```

```
// Stand alone application - find out where the application is
//  running from so that we can pass an absolute path to THIS apps
//   configuration instead of using  the default
//   /usr/local/etc/javabridge/ configuration.
// Note: if you installed the library into a non-standard
//  location (e.g. /opt/test/) and you want a site wide
//  configuration you sould define or set confFile to
// "/opt/test/etc/javabridge/javabridge.conf"

struct stat aFileD;

#if defined(WIN32) || defined(_WIN32)
      #define PATHSEPERATOR ";"
#else
      #define PATHSEPERATOR ":"
#endif  // WIN32

strcpy(confFile, argv[0]);

#if defined(WIN32) || defined(_WIN32)
      // Windows nonsense
      int i;
      for (i=0; i < strlen(confFile); i++)
            if (confFile[i] == '\')
                  confFile[i] = '/';
#endif
if (confFile[0] == '/' || (confFile[1] == ':' && confFile[2] == '/'))
      // called with an absolute invocation
      strcpy(myRoot, confFile);
else
{
      // called with an relative path invocation
      if (strchr(confFile, '/'))
      {
            // called with an relative path invocation
            getcwd(myRoot, sizeof(myRoot));
            strcat(myRoot, "/");
            strcat(myRoot, confFile);
      }
      else
      {
            //check current dir (may not have . on the path)
            getcwd(myRoot, sizeof(myRoot));
            strcat(myRoot, "/");
            strcat(myRoot, confFile);
            if(stat(myRoot, &aFileD) == -1)
            {
                  // called without path, search path to find ourselves
                  char *aPath;
                  char *sysPath = getenv("PATH");
                  if (sysPath != NULL)
                  {
                        aPath = strtok(sysPath, PATHSEPERATOR);
                        while (aPath != NULL)
```

```
                                {
                                      strcpy(myRoot, aPath);
                                      strcat(myRoot, "/");
                                      strcat(myRoot, confFile);
                                      if(stat(myRoot, &aFileD) != -1)
                                            break;
                                      aPath = strtok(NULL, PATHSEPERATOR);
                                }
                                free(sysPath);
                          }
                    }
              }
        }

        *(strrchr(myRoot, '/')) = 0;
        strcpy(confFile, myRoot);
        strcat(confFile, "/javabridge.conf");

        // Information about which .conf file is used
        fprintf(stderr, "ConfFile:%s0, confFile);
        fflush(stderr);

        // We already have something listening on the default port of 9267,
        //  override it here to an unused port
        //  (OK not really - but for examples sake)
        putenv("JAVABRIDGE_LISTENPORT=9267"); //no setenv in MinGW!

        // Start the php-java-bridge listener
        // If its already running, use it otherwise fork a call to start it.

        // Three ways to set conf file location to the library:
        #if defined(USEJENVIRONMENT)
              // Via environment variable
              strcpy(confFile, "JAVABRIDGE_CONFPATH=");
              strcat(confFile, myRoot);
              putenv(confPath);
              javaConnection jvm = startJservice(NULL);
        #else
              // Direct invocation
              javaConnection jvm = startJservice(confFile);
        #endif
        // Or export JAVABRIDGE_CONFPATH in a wrapping shell script
        //  and skip all of the above path nonsense


        // Should now have the JavaBridge listening on 9267
        // Do some stuff with it.

        // Declare something to catch errors with
        //  Note this merely an unsigned int, but JavaBridge.jar keeps
        //   objects as such, and it writes any exception as an instance of
        //   exception, so to work with errors we need to remember what it.
        javaObject anException;
```

```
// Simple test to see if we have something working here

// Construct a Long object.
// Note: the parameters to the constructor which creates the object
//   with value 6
javaObject myLong = constructJobject(jvm, &anException,
      "java.lang.Long", "%d", 6);
// Report errors. Should probably exit(FAIL) here cause nothing
//   further will work if we cannot get past here.
// Note: getException allocates space for the returned string,
//   remember to free it.
if (!myLong)
{ char *e = getJexception(jvm, anException);
      printf("Failed: %s0, e);      free(e);
}
// Get the value of the object.
char *ans = getJstring(jvm, myLong);
if (ans && !strcmp(ans, "6"))
      printf("PASS (getJstring) *** Got:%s Expected ’6’0, ans);
else
      printf("FAIL (getJstring) *** Got:%s Expected ’6’0, ans);
// getString dynamically allocates space for the returned
//   answer, remember to free it.
free(ans);
// Test some other mappings for java values to C values in
//   the library
if (getJlong(jvm, myLong) == 6)
      printf("PASS (getJlong) *** Got:%lld Expected ’6’0,
        getJlong(jvm, myLong));
else
      printf("FAIL (getJlong) *** Got:%lld Expected ’6’0,
        getJlong(jvm, myLong));
if (getJdouble(jvm, myLong) == 6.0)
      printf("PASS (getJdouble) *** Got:%f Expected ’6.0’0,
        getJdouble(jvm, myLong));
else
      printf("FAIL (getJdouble) *** Got:%f Expected ’6.0’0,
        getJdouble(jvm, myLong));
// Release what we have used so that the JVM can do garbage
//    collection.
releaseJobject(jvm, myLong);


// More complex example - do some JDBC stuff.

// Get the class definition for "Class"
//   Note: this is not creating an
//     object (java.lang.Class cannot be instantiated)
javaObject aClass = getJclass(jvm, &anException, "java.lang.Class");
if (!aClass)
      { char *e = getJexception(jvm, anException);
            printf("Failed: %s0, e);
            free(e);
       }
```

```
else
      printf("PASS (getJclass) *** Got:java.lang.Class0);
// Dynamically load a JDBC driver.
//   Note: this is still not creating a java object, simply calling
//   the static method "forName" of the class "Class"
javaObject jdbcDrv = invokeJobject(jvm, &anException, aClass,
  "forName", "%s", "org.sqlite.JDBC");
if (!jdbcDrv)
      { char *e = getJexception(jvm, anException);
            printf("Failed: %s0, e);
            free(e);
      }
else
      printf("PASS (invokeJobject) *** Got:java.lang.Class.forName(org.sqlite.
// We have loaded the driver, get rid of what we used to do so.
releaseJobject(jvm, jdbcDrv);
releaseJobject(jvm, aClass);

// Find the DriverManager class
javaObject manager = getJclass(jvm, &anException,
  "java.sql.DriverManager");
if (!manager)
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (getJclass) *** Got:java.sql.DriverManager0);
// Use DriverManager.getConnection to instantiate our first
//   real java object - a JDBC connection.
javaObject connection = invokeJobject(jvm, &anException, manager,
  "getConnection", "%s", "jdbc:sqlite:testJDBC.sdb");
if (!connection)
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:java.sql.DriverManager.getConnectio
// Do some stuff with the connection object
invokeJobject(jvm, &anException, connection, "setAutoCommit",
  "%b", 1);
// and get a new object from the connection object so that we can
//  do useful stuff
javaObject statement = invokeJobject(jvm, &anException, connection,
  "createStatement", NULL);
if (!statement)
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:Connection.createStatement()0);
// Do some useful stuff
```

```
invokeJobject(jvm, &anException, statement, "execute", "%s",
  "drop table test1");
if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
   "create table test1(col1 char(10), col2 decimal(11.2) )"))
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:Statement.execute(create table)0);
if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
   "insert into test1 values('rowThe1st', 99.99)"))
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:Statement.execute(insert)0);
if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
  "insert into test1 values('rowThe2nd', 0.01)"))
{ char *e = getJexception(jvm, anException);
    printf("Failed: %s0, e);
    free(e);
}
else
      printf("PASS (invokeJobject) *** Got:Statement.execute(insert)0);
// OK, now try and get something out of the data we have written
if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
    "select sum(col2) as tot from test1"))
{ char *e = getJexception(jvm, anException);
      printf("Failed: %s0, e);
      free(e);
}
javaObject resultset = invokeJobject(jvm, &anException, statement,
    "getResultSet", NULL);
if (!statement)
{ char *e = getJexception(jvm, anException);
      printf("Failed: %s0, e);
      free(e);
}
invokeJobject(jvm, &anException, resultset, "next", NULL);
javaObject aString = invokeJobject(jvm, &anException, resultset,
  "getString", "%s", "tot");
if (getJdouble(jvm, aString) == 100.0)
      printf("PASS (jdbc)*** Got:%s Expected 100.000,
        getJstring(jvm, aString));
else
      printf("FAIL (jdbc)*** Got:%s Expected 100.000,
        getJstring(jvm, aString));
// Release what we have used so the JVM can do garbage collection
releaseJobject(jvm, aString);
// Close our result set and allow the JVM to garbage collect it
invokeJobject(jvm, &anException, resultset, "close", NULL);
releaseJobject(jvm, resultset);
```

```
        // Test that we are in fact getting exceptions correctly.
        if (!invokeJobject(jvm, &anException, statement, "execute", "%s",
          "create table test1(col1 char(10), col2 decimal(11.2) )"))
        {
              char *e = getJexception(jvm, anException);
              printf("PASS (exception)*** Got:%s 0, e);
              printf("  Expected ’SQLException: table test1 already exists’0);
              free(e);
        }
        else
        {
              printf("FAIL (exception)*** Got:Success ");
              printf("Expected ’SQLException: table test1 already exists’0);
        }
        // Clean up
        invokeJobject(jvm, &anException, statement, "close", NULL);
        releaseJobject(jvm, statement);
        invokeJobject(jvm, &anException, connection, "close", NULL);
        releaseJobject(jvm, connection);
        releaseJobject(jvm, manager);


        // And drop our connection to the running application server
        //   Note this doesn’t really stop the server running, it simply
        //    tells the server that we are not interested in further
        //    communication, and that it can clean up, close sockets and etc.
        stopJservice(jvm);

        // OR if you really don’t need the service anymore do an ugly thing
        // and kill it dead.
        //   Note this really stops the server running in an unclean way,
        //    and anything opened by the server will be dropped where it is.

        // Cause I am testing & testing & testing, I don’t want the service
        //    left running.

        // Got to get a new connections, as it it is dropped above.
        jvm = startJservice(confFile);
        // And kill it
        killJservice(jvm);

  }
```

SEE ALSO

http://php-java-bridge.sf.net

REPORTING PROBLEMS

I am a long way from being fluent in java. If your issue is with java consult the java api docs, or look at oracles web site.

There is *javabridgeservice.bat* (for windows) and *javabridgeservice.sh* (for unix like OS) in install_path/etc. These scripts allow interactivly starting JavaBridge.jar with debugging turned on from a console. You may get a better idea of where things are failing by inspecting the output here while running your code.

php-java-bridge has a mailing list for users and comprehensive documentation for their bridge on source forge.

If you are certain that you have found a bug in the C library, *dirk@ddtdebuggers.co.za*

AUTHORS

Version 0.9, 2015/05/05

This software interfaces with php-java-bridge provided at *http://php-java-bridge.sf.net* While this library is not afilliated to that group, it is useless without the package that they supply. To my best understanding *php-java-bridge* is licensed for all use without restriction.

My deepest appreciation goes to the php-java-bridge team.