GNU Linear Programming Kit
Java Binding

Reference Manual

Version 1.0.11

March 2010

Contents

Chapter 1

Introduction

The GNU Linear Programming Kit (GLPK) package supplies a solver for large scale linear programming (LP) and mixed integer programming (MIP). The GLPK project is hosted at http://www.gnu.org/software/glpk.

It has two mailing lists:

- help-glpk@gnu.org and
- bug-glpk@gnu.org.

To subscribe to one of these lists, please, send an empty mail with a Subject: header line of just "subscribe" to the list.

GLPK provides a library written in C and a standalone solver.

The source code provided at ftp://gnu.ftp.org/gnu/glpk/ contains the documentation of the library in file doc/glpk.pdf.

The Java platform provides the Java Native Interface (JNI) to integrate non-Java language libraries into Java applications.

Project GLPK-Java delivers a Java Binding for GLPK. It is hosted at http://glpk-java.sourceforge.net/.

To report problems and suggestions concerning GLPK-Java, please, send an email to the author at xypron.glpk@gmx.de.

Chapter 2

Architecture

A GLPK-Java application will consist of the following

- the GLPK library
- the GLPK-Java JNI library
- the GLPK-Java class library
- the application code.

2.1 GLPK library

2.1.1 Source

The source code to compile the GLPK library is provided at ftp://gnu.ftp.org/gnu/glpk/.

2.1.2 Linux

The GLPK library can be compiled from source code. Follow the instructions in file INSTALL provided in the source distribution. Precompiled packages are available in many Linux distributions.

The usual installation path for the library is /usr/local/lib/libglpk.so.

2.1.3 Windows

The GLPK library can be compiled from source code. The build and make files are in directory w32 for 32 bit Windows and in w64 for 64 bit Windows. The name of the created library is glpk_4_43.dll for revision 4.43.

A precompiled version of GLPK is provided at http://winglpk.sourceforge.net.

The library has to be in the search path for binaries. Either copy the library to a directory that is already in the path (e.g. C:\windows\system32) or update the path in the system settings of Windows.

2.2 GLPK-Java JNI library

2.2.1 Source

The source code to compile the GLPK-Java JNI library is provided at http://glpk-java.sourceforge.net.

2.2.2 Linux

The GLPK-Java JNI library can be compiled from source code. Follow the instructions in file INSTALL provided in the source distribution.

The usual installation path for the library is /usr/local/lib/libglpk-java.so.

2.2.3 Windows

The GLPK-Java JNI library can be compiled from source code. The build and make files are in directory w32 for 32 bit Windows and in w64 for 64 bit Windows. The name of the created library is glpk_4_43_java.dll for revision 4.43.

A precompiled version of GLPK-Java is provided at http://winglpk.sourceforge.net.

The library has to be in the search path for binaries. Either copy the library to a directory that is already in the path (e.g. C:\windows\system32) or update the path in the system settings of Windows.

2.3 GLPK-Java class library

The source code to compile the GLPK-Java class library is provided at http://glpk-java.sourceforge.net.

2.3.1 Linux

The GLPK-Java class library can be compiled from source code. Follow the instructions in file INSTALL provided in the source distribution.

The usual installation path for the library is /usr/local/share/java/glpk-java.jar.

For Debian and Ubuntu the following packages are needed for compilation:

- libtool
- swig
- java-gcj-compat-dev

2.3.2 Windows

The GLPK-Java class library can be compiled from source code. The build and make files are in directory w32 for 32 bit Windows and in w64 for 64 bit Windows. The name of the created library is glpk-java.jar.

A precompiled version of GLPK including GLPK-Java is provided at http://winglpk.sourceforge.net.

The library has to be in the CLASSPATH. Update the classpath in the system settings of Windows or specify the classpath upon invocation of the application, e.g.

```
java -classpath ./glpk-java.jar;. MyApplication
```

Chapter 3

Classes

GLPK-Java uses the Simplified Wrapper and Interface Generator (SWIG) to create the JNI interface to GLPK. Classes are created in path org.gnu.glpk.

Interface GlpkCallbackListener can be implemented to register a listener for class GlpkCallback.

Class GlpkCallback is called by the MIP solver callback routine.

Class GlpkException is thrown if an error occurs.

Class GLPK maps the functions from include/glpk.h.

Class GLPKConstants maps the constants from include/glpk.h to methods.

Class GLPKJNI contains the definitions of the native functions.

The following classes map structures from include/glpk.h:

- glp_attr
- glp_bfcp
- glp_cpxcp
- glp_data
- glp_iocp
- glp_iptcp
- glp_long

- glp_mpscp
- glp_prob
- glp_smcp
- glp_tran
- glp_tree
- LPXKKT
- _glp_arc
- _glp_graph
- _glp_vertex

The following classes are used to map pointers:

- SWIGTYPE_p_double
- SWIGTYPE_p_f_p_glp_tree_p_void__void
- SWIGTYPE_p_f_p_void_p_q_const__char__int
- SWIGTYPE_p_glp_bfcp
- SWIGTYPE_p_int
- SWIGTYPE_p_p_char
- SWIGTYPE_p_p__glp_vertex
- SWIGTYPE_p_va_list
- SWIGTYPE_p_void

Chapter 4

Usage

4.1 Exceptions

When illegal parameters are passed to a function of the GLPK native library an exception GlpkException is thrown. Due to the architecture of GLPK all GLPK objects are invalid when such an exception has occured.

4.1.1 Implementation details

GLPK Java registers a function glp_java_error_hook() to glp_error_hook() before calling an GLPK API function. If an error occurs function glp_free_env is called and a long jump is used to return to the calling environment. Then function glp_java_throw() is called which throws GlpkException.

4.2 Callbacks

The MIP solver provides a callback functionality. This is used to call method callback of class GlpkCallback. A Java program can listen to the callbacks by instantiating a class implementing interface GlpkCallbackListener and registering the object with method addListener() of class GlpkCallback. The listener can be deregistered with method removeListener(). The listener can use method GLPK.glp_ios_reason() to find out why it is called. For details see the GLPK library documentation.

4.3 Loading JNI library

To be able to use the JNI library in a Java program it has to be loaded. The following loading code is used in class GLPK.

```
try
{
    // try to load Linux library
    System.loadLibrary("glpk_java");
}
catch (UnsatisfiedLinkError e)
{
    // try to load Windows library
    System.loadLibrary("glpk_4_43_java");
}
```

If the JNI library could not be loaded, you will receive an exception java.lang.UnsatisfiedLinkError.

On a Linux system the exception message will return the Windows library name though glpk_java.so could not be loaded.

Chapter 5

Examples

Examples are provided in directory examples/java of the source distribution of GLPK-Java.

To compile the examples the classpath must point to glpk-java.jar, e.g.

```
javac -classpath /usr/local/shared/java/glpk-java.jar Example.java
```

To run the examples the classpath must point to glpk-java.jar. The java.library.path must point to the directory with the dynamic link libraries, eg.

```
java -Djava.library.path=/usr/local/lib \
-classpath /usr/local/shared/java/glpk-java.jar:. \
Example
```

5.1 Lp.java

5.1.1 Description

This example solves a small linear problem and ouputs the solution.

5.1.2 Coding

```
import org.gnu.glpk.GLPK;
import org.gnu.glpk.GLPKConstants;
import org.gnu.glpk.GlpkException;
import org.gnu.glpk.SWIGTYPE_p_double;
```

```
import org.gnu.glpk.SWIGTYPE_p_int;
import org.gnu.glpk.glp_prob;
import org.gnu.glpk.glp_smcp;

public class Lp {
    // Minimize z = (x1-x2) /2 + (1-(x1-x2)) = -.5 * x1 + .5 * x2 + 1
    //
    // subject to
    // 0.0<= x1 - x2 <= 0.2
    // where,
    // 0.0 <= x1 <= 0.5
    // 0.0 <= x2 <= 0.5

    public static void main(String[] arg) {
        glp_prob lp;
        glp_smcp parm;
        SWIGTYPE_p_int ind;
        SWIGTYPE_p_double val;
        int ret;

        try {
            // Create problem
            lp = GLPK.glp_create_prob();
            System.out.println("Problem created");
            GLPK.glp_set_prob_name(lp, "myProblem");

            // Define columns
            GLPK.glp_add_cols(lp, 2);
            GLPK.glp_set_col_name(lp, 1, "x1");
            GLPK.glp_set_col_kind(lp, 1, GLPKConstants.GLP_CV);
            GLPK.glp_set_col_bnds(lp, 1, GLPKConstants.GLP_DB, 0, .5);
            GLPK.glp_set_col_name(lp, 2, "x2");
            GLPK.glp_set_col_kind(lp, 2, GLPKConstants.GLP_CV);
            GLPK.glp_set_col_bnds(lp, 2, GLPKConstants.GLP_DB, 0, .5);

            // Create constraints
            GLPK.glp_add_rows(lp, 1);

            GLPK.glp_set_row_name(lp, 1, "c1");
            GLPK.glp_set_row_bnds(lp, 1, GLPKConstants.GLP_DB, 0, 0.2);
            ind = GLPK.new_intArray(3);
            GLPK.intArray_setitem(ind, 1, 1);
            GLPK.intArray_setitem(ind, 2, 2);
            val = GLPK.new_doubleArray(3);
```

```
            GLPK.doubleArray_setitem(val, 1, 1.);
            GLPK.doubleArray_setitem(val, 2, -1.);
            GLPK.glp_set_mat_row(lp, 1, 2, ind, val);

            // Define objective
            GLPK.glp_set_obj_name(lp, "z");
            GLPK.glp_set_obj_dir(lp, GLPKConstants.GLP_MIN);
            GLPK.glp_set_obj_coef(lp, 0, 1.);
            GLPK.glp_set_obj_coef(lp, 1, -.5);
            GLPK.glp_set_obj_coef(lp, 2, .5);

            // Solve model
            parm = new glp_smcp();
            GLPK.glp_init_smcp(parm);
            ret = GLPK.glp_simplex(lp, parm);

            // Retrieve solution
            if (ret == 0) {
                write_lp_solution(lp);
            } else {
                System.out.println("The problem could not be solved");
            }

            // Free memory
            GLPK.glp_delete_prob(lp);
        } catch (GlpkException ex) {
            ex.printStackTrace();
        }
    }

    /**
     * write simplex solution
     * @param lp problem
     */
    static void write_lp_solution(glp_prob lp) {
        int i;
        int n;
        String name;
        double val;

        name = GLPK.glp_get_obj_name(lp);
        val = GLPK.glp_get_obj_val(lp);
        System.out.print(name);
        System.out.print(" = ");
```

```
        System.out.println(val);
        n = GLPK.glp_get_num_cols(lp);
        for (i = 1; i <= n; i++) {
            name = GLPK.glp_get_col_name(lp, i);
            val = GLPK.glp_get_col_prim(lp, i);
            System.out.print(name);
            System.out.print(" = ");
            System.out.println(val);
        }
    }
}
```

5.2 Gmpl.java

5.2.1 Description

This example reads a GMPL file and executes it. The callback function is used to write an output line when a better MIP soluton has been found.

Run the program with the model file as parameter.

```
java -Djava.library.path=/usr/local/lib \
-classpath /usr/local/shared/java/glpk-java.jar:. \
GLPKSwig marbles.mod
```

5.2.2 Coding

```
import org.gnu.glpk.GLPK;
import org.gnu.glpk.GLPKConstants;
import org.gnu.glpk.GlpkCallback;
import org.gnu.glpk.GlpkCallbackListener;
import org.gnu.glpk.glp_iocp;
import org.gnu.glpk.glp_prob;
import org.gnu.glpk.glp_tran;
import org.gnu.glpk.glp_tree;

public class Gmpl implements GlpkCallbackListener {

    public static void main(String[] arg) {
        if (1 != arg.length) {
            System.out.println("Usage: java Gmpl model.mod");
```

```
        return;
    }
    new Gmpl().solve(arg);
}

public void solve(String[] arg) {
    glp_prob lp = null;
    glp_tran tran;
    glp_iocp iocp;

    String fname;
    int skip = 0;
    int ret;

    GlpkCallback.addListener(this);

    fname = new String(arg[0]);

    lp = GLPK.glp_create_prob();
    System.out.println("Problem created");

    tran = GLPK.glp_mpl_alloc_wksp();
    ret = GLPK.glp_mpl_read_model(tran, fname, skip);
    if (ret != 0) {
        GLPK.glp_mpl_free_wksp(tran);
        GLPK.glp_delete_prob(lp);
        throw new RuntimeException("Model file not found: " + fname);
    }

    // generate model
    GLPK.glp_mpl_generate(tran, null);
    // build model
    GLPK.glp_mpl_build_prob(tran, lp);
    // set solver parameters
    iocp = new glp_iocp();
    GLPK.glp_init_iocp(iocp);
    iocp.setPresolve(GLPKConstants.GLP_ON);
    // solve model
    ret = GLPK.glp_intopt(lp, iocp);
    // postsolve model
    if (ret == 0) {
        GLPK.glp_mpl_postsolve(tran, lp, GLPKConstants.GLP_MIP);
    }
    // free memory
```

```
        GLPK.glp_mpl_free_wksp(tran);
        GLPK.glp_delete_prob(lp);
    }

    public void callback(glp_tree tree) {
        int reason = GLPK.glp_ios_reason(tree);
        if (reason == GLPKConstants.GLP_IBINGO) {
            System.out.println("Better solution found");
        }
    }
}
```