GNU Linear Programming Kit
Java Binding

Reference Manual

Version 1.0.28

May 30, 2013

Contents

Chapter 1

Introduction

The GNU Linear Programming Kit (GLPK)[2] 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)[3] to integrate non-Java language libraries into Java applications.

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

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

Chapter 2

Getting started

This chapter will run you through the installation of GLPK for Java and the execution of a trivial example.

2.1 Installation

2.1.1 Windows

The following description assumes:

- You are using a 64-bit version of Windows. Replace folder name w64 by w32 if you are using a 32-bit version.
- The current version of GLPK is 4.50. Please adjust pathes if necessary.
- Your path for program files is "C:\Program Files". Please adjust pathes if necessary.

Download the current version of GLPK for Windows from https://sourceforge.net/projects/winglpk/.

The filename for version 4.50 is winglpk-4.50.zip. Unzip the file. Copy folder glpk-4.50 to "C:\Program Files\GLPK\".

To check the installation run the following command:

```
"C:\Program Files\GLPK\w64\glpsol.exe" --version
```

To use GLPK for Java you need a Java development kit to be installed. The Oracle JDK can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/index.html.

To check the installation run the following commands:

```
"%JAVA_HOME%\bin\javac" -version
java -version
```

2.1.2 Linux

Debian package

For Debian and Ubuntu an installation package for GLPK for Java exists. It can be installed by the following commands:

```
sudo apt-get install libglpk-java
```

The installation will be in /usr not in /usr/local as assumed in the examples below.

Installation from source

Download the current version of GLPK source with

```
wget ftp://ftp.gnu.org/gnu/glpk/glpk-4.50.tar.gz
```

Unzip the archive with:

```
tar -xzf glpk-4.50.tar.gz
cd glpk-4.50
```

Configure with

```
./configure
```

Make and install with:

```
make
make check
sudo make install
```

Check the installation with

```
glpsol --version
```

For the next steps you will need a Java Development Kit (JDK) to be installed.

You can check the correct installation with the followng commands:

```
$JAVA_HOME/bin/javac -version
java -version
```

If the JDK is missing refer to http://openjdk.java.net/install/ for installation instructions.

To build GLPK for Java you will need package SWIG (Simplified Wrapper and Interface Generator, http://www.swig.org/). You can check the installation with the following command:

```
swig -version
```

Most Linux distribution contain a SWIG package. The installation command will depend on the distribution, e.g.

- Debian: sudo apt-get install swig
- Fedora: sudo yum install swig

- Gentoo: sudo emerge swig

Download GLPK for Java from https://sourceforge.net/projects/glpk-java/files/.

Unzip the archive with:

```
tar -xzf glpk-java-1.0.28.tar.gz
cd glpk-java-1.0.28
```

Configure with:

```
./configure
```

OS X has jni.h in a special path. You may want to specify this path in the parameters CPPFLAGS and SWIGFLAGS for the configure script

```
./configure \
  CPPFLAGS=-I/System/Library/Frameworks/JavaVM.framework/Headers \
  SWIGFLAGS=-I/System/Library/Frameworks/JavaVM.framework/Headers
```

If libglpk.so is in a special path you may specify this path using parameter LDFLAGS, e.g.

```
./configure LDFLAGS=-L/opt/lib
```

Make and install with:

```
make
Make and install with:

\begin{lstlisting}
make
make check
sudo make install
```

2.2 Trivial example

In the example we will create a Java class which will write the GLPK version to the console.

With a text editor create a text file Test.java with the following content:

```
import org.gnu.glpk.GLPK;
public class Test {
  public static void main(String[] args) {
    System.out.println( GLPK.glp_version());
  }
}
```

2.2.1 Windows

Compile the class

```
set CLASSPATH=C:Program Files\GLPK\glpk-4.50\w64\glpk-java.jar
"%JAVA_HOME%/bin/javac" Test.java
```

Run the class

```
set CLASSPATH=C:\Program Files\GLPK\glpk-4.50\w64\glpk-java.jar;.
java -Djava.library.path="C:Program Files\GLPK\glpk-4.50\w64" Test
```

The output will be the GLPK version number, for example: 4.50.

2.2.2 Linux

Compile the class

```
javac -classpath /usr/local/share/java/glpk-java.jar Test.java
```

Run the class:

```
java -Djava.library.path=/usr/local/lib/jni \
-classpath /usr/local/share/java/glpk-java.jar:. \
Test
```

The output will be the GLPK version number, for example: 4.50.

Chapter 3

Architecture

A GLPK for Java application will consist of

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

3.1 GLPK library

3.1.1 Source

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

3.1.2 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.

3.1.3 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_50.dll for revision 4.50.

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.

3.2 GLPK for Java JNI library

3.2.1 Source

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

3.2.2 Linux

The GLPK for 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.

3.2.3 Windows

The GLPK for 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_50_java.dll for revision 4.50.

A precompiled version of GLPK for 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.

3.3 GLPK for Java class library

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

3.3.1 Linux

The GLPK for 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
- openjdk-6-jdk (or a higher version)

3.3.2 Windows

The GLPK for 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.

3.3.3 Classpath

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 4

Maven

For using this library in your Maven project enter the following repository and dependency in your pom.xml:

```
<repositories>
    <repository>
        <id>XypronRelease</id>
        <name>Xypron Release</name>
        <url>http://rsync.xypron.de/repository</url>
        <layout>default</layout>
    </repository>
</repositories>

<dependencies>
    <dependency>
        <groupId>org.gnu.glpk</groupId>
        <artifactId>glpk-java</artifactId>
        <version>1.0.28</version>
    </dependency>
</dependencies>
```

The artifact does not include the binary libraries, which have to be installed separately.

Chapter 5

Classes

GLPK for Java uses the Simplified Wrapper and Interface Generator (SWIG)[4] to create the JNI interface to GLPK. Classes are created in path org.gnu.glpk.

Class GlpkCallback is called by the MIP solver callback routine.

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

Class GlpkTerminal is called by the MIP solver terminal output routine.

Interface GlpkTerminalListener can be implemented to register a listener for class GlpkTerminal.

Class GlpkException is thrown if an error occurs.

Class GLPK maps the functions from glpk.h.

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

Class GLPKJNI contains the definitions of the native functions.

The following classes map structures from glpk.h:

- glp_attr
- glp_bfcp
- glp_cpxcp
- 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_q_const__char_v_____void
- SWIGTYPE_p_f_p_void__void
- SWIGTYPE_p_f_p_void_p_q_const__char__int
- SWIGTYPE_p_int
- SWIGTYPE_p_glp_arc
- SWIGTYPE_p_glp_graph
- SWIGTYPE_p_glp_vertex
- SWIGTYPE_p_va_list
- SWIGTYPE_p_void

The following clases are used for network problems:

- _glp_java_arc_data
- _glp_java_vertex_data

Chapter 6

Usage

Please, refer to file doc/glpk.pdf of the GLPK source distribution for a detailed description of the methods and constants.

6.1 Loading the JNI library

To be able to use the JNI library in a Java program it has to be loaded. The path to dynamic link libaries can specified on the command line when calling the Java runtime, e.g.

```
java -Djava.library.path=/usr/local/lib/jni/libglpk_java
```

The following code is used in class GLPKJNI to load the JNI library:

```
static {
  try {
    if (System.getProperty("os.name").toLowerCase().contains("windows")) {
      // try to load Windows library
      try {
        System.loadLibrary("glpk_4_50");
      } catch (UnsatisfiedLinkError e) {
        // The dependent library might be in the OS library search path.
      }
      System.loadLibrary("glpk_4_50_java");
    } else {
      // try to load Linux library
      System.loadLibrary("glpk_java");
    }
  } catch (UnsatisfiedLinkError e) {
    System.err.println(
      "The dynamic link library for GLPK for Java could not be"
      + "loaded.\nConsider using\njava -Djava.library.path=");
    throw e;
  }
}
```

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

6.2 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.

6.2.1 Implementation details

GLPK for 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.

6.3 Network problems

For network problems additional data like capacity and cost of arcs or the inflow of vertics has to be specified. The GLPK library does not provide data structures. In GLPK for Java classes _glp_java_arc_data and _glp_java_vertex_data are provided.

When creating a graph the size of the structures for these classes has to be specified. In some routines the offsets to individual fields in the structures are needed. The following constants have been defined:

- GLP_JAVA_A_CAP - offset of field cap in arc data
- GLP_JAVA_A_COST - offset of field cost in arc data
- GLP_JAVA_A_LOW - offset of field low in arc data
- GLP_JAVA_A_RC - offset of field rc in arc data
- GLP_JAVA_A_X - offset of field x in arc data
- GLP_JAVA_A_SIZE - size of arc data
- GLP_JAVA_V_CUT - offset of field cut in vertex data
- GLP_JAVA_V_PI - offset of field pi in vertex data
- GLP_JAVA_V_RHS - offset of field rhs in vertex data
- GLP_JAVA_V_SET - offset of field set in vertex data
- GLP_JAVA_V_SIZE - size of vertex data

For accessing vertices method GLPK.glp_java_vertex_get can be used.

For accessing the data areas of arcs and vertices methods GLPK.glp_java_arc_get_data, GLPK.glp_java_vertex_data_get, and GLPK.glp_java_vertex_get_data can be used.

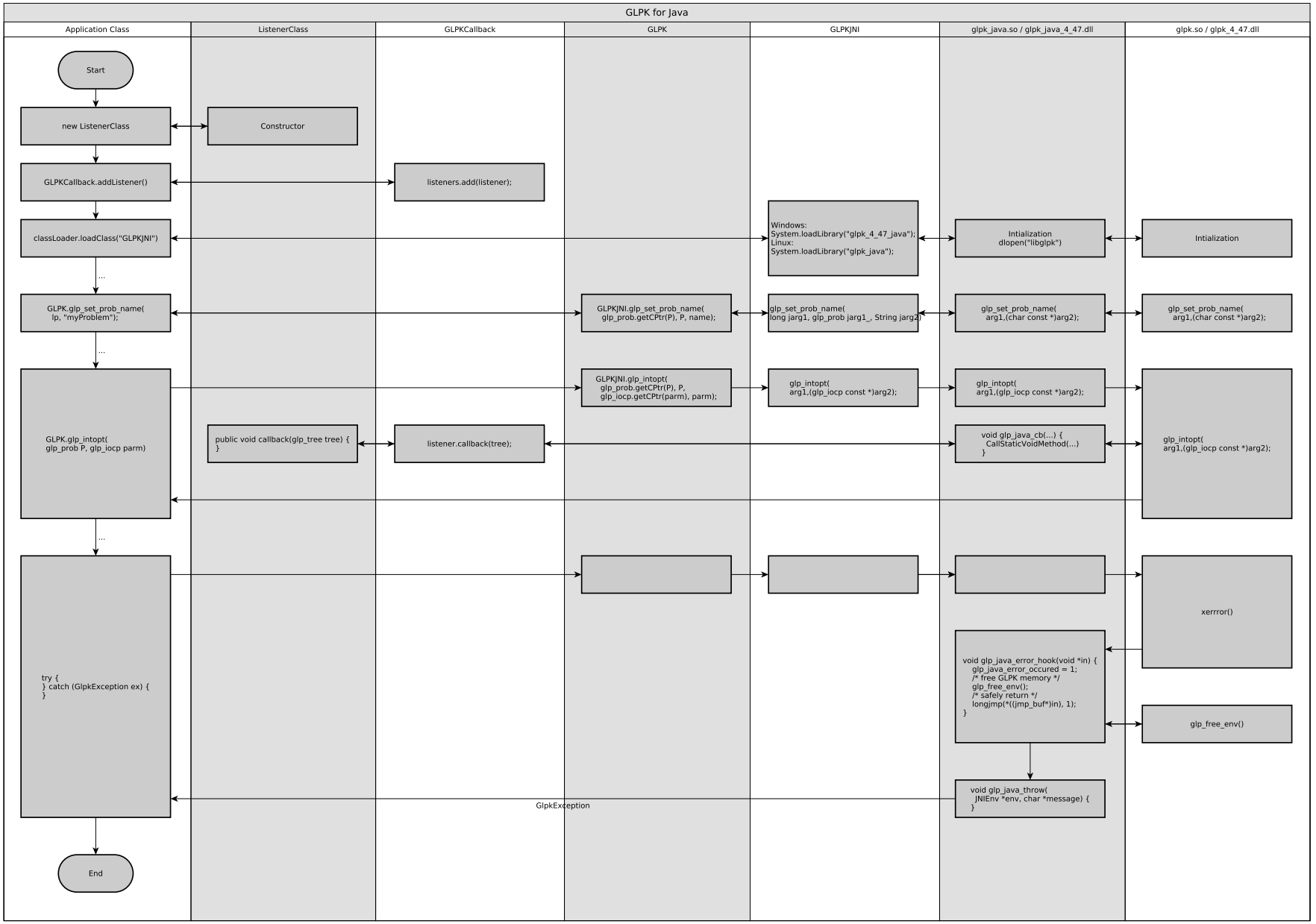
```
glp_arc arc;
_glp_java_arc_data adata;
_glp_java_vertex_data vdata;

glp_graph graph =
        GLPK.glp_create_graph(
```

```
        GLPKConstants.GLP_JAVA_V_SIZE,
        GLPKConstants.GLP_JAVA_A_SIZE);
GLPK.glp_set_graph_name(graph,
        MinimumCostFlow.class.getName());

int ret = GLPK.glp_add_vertices(graph, 9);

GLPK.glp_set_vertex_name(graph, 1, "v1");
GLPK.glp_set_vertex_name(graph, 2, "v2");
GLPK.glp_set_vertex_name(graph, 3, "v3");
GLPK.glp_set_vertex_name(graph, 4, "v4");
GLPK.glp_set_vertex_name(graph, 5, "v5");
GLPK.glp_set_vertex_name(graph, 6, "v6");
GLPK.glp_set_vertex_name(graph, 7, "v7");
GLPK.glp_set_vertex_name(graph, 8, "v8");
GLPK.glp_set_vertex_name(graph, 9, "v9");

vdata = GLPK.glp_java_vertex_data_get(graph, 1);
vdata.setRhs(20);
vdata = GLPK.glp_java_vertex_data_get(graph, 9);
vdata.setRhs(-20);

arc = GLPK.glp_add_arc(graph, 1, 2);
adata = GLPK.glp_java_arc_get_data(arc);
adata.setLow(0); adata.setCap(14); adata.setCost(0);

...

GLPK.glp_write_mincost(graph,
        GLPKConstants.GLP_JAVA_V_RHS,
        GLPKConstants.GLP_JAVA_A_LOW,
        GLPKConstants.GLP_JAVA_A_CAP,
        GLPKConstants.GLP_JAVA_A_COST,
        "mincost.dimacs");
GLPK.glp_delete_graph(graph);
```

6.4 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.

Figure 6.1: Callbacks and Error Handling



6.5 Output listener

GLPK provides a hook for terminal output. A Java program can listen to the callbacks by instantiating a class implementing interface GlpkTerminalListener and registering the object with method addListener of class GlpkTerminal. The listener can be dregistered with method removeListener(). After a call to glp_free_env() the GlpkTerminal has to registered again by calling GLPK.glp_term_hook(null, null). glp_free_env() is called if an exception GlpkException occurs.

6.6 Aborting a GLPK library call

Method void GLPK.glp_java_error(String message) can be used to abort any call to the GLPK library. An exception GlpkException will occur. As GLPK is not threadsafe the call must be placed in the same thread as the initial call that is to be aborted. The output method of a GlpkTerminalListener can be used for this purpose.

6.7 Debugging support

Method void GLPK.glp_java_set_msg_lvl(int msg_lvl) can be used to enable extra output signaling when a GLPK library function is entered or left using value with GLPKConstants.GLP_JAVA_MSG_LVL_ALL. The output is disabled by a call with value GLPKConstants.GLP_JAVA_MSG_LVL_OFF.

6.8 Locales

Method void GLPK.glp_java_set_numeric_locale(String locale) can be used to set the locale for numeric formatting. When importing model files the GLPK library expects to be using locale "C".

6.9 Threads

The GLPK library is not thread safe. Never two threads should be running that access the GLPK library at the same time. When a new thread accesses the library it should call GLPK.glp_free_env(). When using an GlpkTerminalListener it is necessary to register GlpkTerminal again by calling GLPK.glp_term_hook(null, null).

When writing a GUI application it is advisable to use a separate thread for the calls to GLPK. Otherwise the GUI cannot react to events during the call to the GLPK libary.

Chapter 7

Examples

Examples are provided in directory examples/java of the source distribution of GLPK for 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, e.g.

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

7.1 Lp.java

7.1.1 Description

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

7.1.2 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);
        GLPK.delete_intArray(ind);
        GLPK.delete_doubleArray(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);
        }
    }
}
```

7.2 Gmpl.java

7.2.1 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
```

7.2.2 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");
        }
    }
}
```

Chapter 8

License

GLPK for Java is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License[1] as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GLPK for Java is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GLPK for Java. If not, see http://www.gnu.org/licenses/.

Bibliography

[1] Free Software Foundation, Inc. *GNU General Public License*, 2007.

[2] Andrew Makhorin. *GNU Linear Programming Kit*. GNU Software Foundation, 2010.

[3] Sun Microsystems, Inc. *Java Native Interface Specification v1.5*, 2004.

[4] SWIG.org. *Simplified Wrapper and Interface Generator*, 2010.

Index