Creating Xpress Solver applications

47.01

QUICK REFERENCE

FICO® Xpress Optimization

FICO

©2023–2026 Fair Isaac Corporation. All rights reserved. This documentation is the property of Fair Isaac Corporation ("FICO"). Receipt or possession of this documentation does not convey rights to disclose, reproduce, make derivative works, use, or allow others to use it except solely for internal evaluation purposes to determine whether to purchase a license to the software described in this documentation, or as otherwise set forth in a written software license agreement between you and FICO (or a FICO affiliate). Use of this documentation and the software described in it must conform strictly to the foregoing permitted uses, and no other use is permitted.

The information in this documentation is subject to change without notice. If you find any problems in this documentation, please report them to us in writing. Neither FICO nor its affiliates warrant that this documentation is error-free, nor are there any other warranties with respect to the documentation except as may be provided in the license agreement. FICO and its affiliates specifically disclaim any warranties, express or implied, including, but not limited to, non-infringement, merchantability and fitness for a particular purpose. Portions of this documentation and the software described in it may contain copyright of various authors and may be licensed under certain third-party licenses identified in the software, documentation, or both.

In no event shall FICO or its affiliates be liable to any person for direct, indirect, special, incidental, or consequential damages, including lost profits, arising out of the use of this documentation or the software described in it, even if FICO or its affiliates have been advised of the possibility of such damage. FICO and its affiliates have no obligation to provide maintenance, support, updates, enhancements, or modifications except as required to licensed users under a license agreement.

FICO is a registered trademark of Fair Isaac Corporation in the United States and may be a registered trademark of Fair Isaac Corporation in other countries. Other product and company names herein may be trademarks of their respective owners.

Patent(s): www.fico.com/en/patents

FICO® Xpress Optimization 9.9

Deliverable Version: A

Last Revised: 15 May, 2026

Contents

Introduction 1

Prerequisites . . . 1

1 Creating Xpress Solver Applications in C or C++ 2

1.1 Windows . . . 3
1.1.1 Creating an application with Visual Studio . . . 4
1.2 Unix . . . 5
1.2.1 Configuring the environment to find Xpress libraries . . . 5
1.2.2 Hard coding the library location into the binary . . . 5
1.2.3 Creating an application with Xcode . . . 6
1.3 Example makefiles . . . 7
1.3.1 Linking with the static Xpress library (Linux) . . . 7
1.4 Create Xpress Solver Applications in Eclipse . . . 7

2 Creating Xpress Solver Applications in Java 9

2.1 Compiling source files that use the Xpress Solver . . . 10
2.2 Running an application that uses the Xpress Solver . . . 10
2.3 Setting up Eclipse for Xpress . . . 11

3 Creating Xpress Solver Applications in Csharp 13

3.1 Creating Xpress applications in Visual Studio . . . 14
3.1.1 Using NuGet packages . . . 14

Appendix 16

A Contacting FICO 16

FICO Customer Support . . . 16
Documentation . . . 16
FICO Learning . . . 17
Sales and maintenance . . . 17
About FICO . . . 17

Introduction

In this document we describe how to create applications that use the Xpress Solver as a library. We don't focus on the actual content or purpose of the application. Instead we focus on how to create the application binaries from the source code.

The document covers several programming languages that are supported by the Xpress Solver. The document is limited to programming languages/libraries that are directly supported and shipped by FICO. There are many other programming languages that support the Xpress Solver but are supported by 3rd party maintainers.

The document uses the same application for all programming languages: a code that solves the simple optimization problem

$$
\begin{aligned}
&\text{minimize} \\
&\qquad 3x_1 + 5x_2 \\
&\text{subject to} \\
&\qquad 2x_1 + x_2 \geqslant 3 \\
&\qquad 2x_1 + 2x_2 \geqslant 5 \\
&\qquad x_1 + 4_2 \geqslant 4 \\
&\qquad x_1,\ x_2 \geqslant 0
\end{aligned}
$$

More elaborate application examples can be found in the `examples` directory of your Xpress installation.

Prerequisites

All the instructions provided here assume that you have installed Xpress into a folder called `C:\xpressmp` on Windows and `/xpressmp` on other platforms. If your installation folder has a different name then you have to replace it appropriately with the full absolute path of your installation folder. Note that if the path to your installation contains blanks or other whitespace, then you need to quote that appropriately in almost all cases.

Furthermore we assume that your environment is configured so that an Xpress license can be found in a default location. Such a configuration is typically done by having the `XPAUTH_PATH` environment variable point to the `xpauth.xpr` file that contains your license.
On Windows, another way is to put the `xpauth.xpr` file next to the Xpress library, since that is one of the default locations in which Xpress looks for the license.

CHAPTER 1

Creating Xpress Solver Applications in C or C++

When creating an application that is written in the C or C++ programming language, there are two steps:

1. Compile the source code into an object file.
2. Link the object file and the Xpress library into an executable.

Both steps are rather different on Windows and Unix platforms, so things are explained in two different sections.

We will give instructions for compiling and linking a C application. Creating an application written in C++ is basically the same and only requires some additional settings. We will specify these settings where necessary.

In the below we assume that you have code in a file called app.c. An example could be this

```
/* A very simple Xpress application that solves this optimization problem:
 *    Minimize
 *      3x1 + 5x2
 *    Subject To
 *      2x1 +  x2 >= 3
 *      2x1 + 2x2 >= 5
 *       x1 + 4x2 >= 4
 *      x1,x2 >= 0
 * and displays the results.
 * For more detailed examples please see the examples/solver/optimizer/c
 * directory in your Xpress installation.
 */
#include <xprs.h>
#include <stdio.h>

int
main(void)
{
  double obj[] = { 3, 5 };
  double lb[] = { 0, 0 };
  double ub[] = { XPRS_PLUSINFINITY, XPRS_PLUSINFINITY };
  double rhs[] = { 3, 5, 4 };
  int rowind[] = { 0, 1, 2, 0, 1, 2 };
  double rowcoef[] = { 2, 2, 1, 1, 2, 4 };
  int start[] = { 0, 3, 6 };
  double x[2];
  XPRSprob prob;

  if ( XPRSinit("") ) {
    char buffer[256];
    XPRSgetlicerrmsg(buffer, sizeof(buffer));
    fprintf(stderr, "License not found: %s\n", buffer);
    return -1;
  }
  if ( XPRScreateprob(&prob) )
    return -1;
```

```
  if ( XPRSloadlp(prob, "problem", 2, 3, "GGG", rhs, NULL, obj, start, NULL,
                  rowind, rowcoef, lb, ub) )
    return -1;
  if ( XPRSaddnames(prob, XPRS_NAMES_COLUMN, "x1\0x2", 0, 1) )
    return -1;

  if ( XPRSoptimize(prob, "", NULL, NULL) )
    return -1;

  if ( XPRSgetsolution(prob, NULL, x, 0, 1) )
    return -1;

  printf("x1 = %f\n", x[0]);
  printf("x2 = %f\n", x[1]);

  XPRSdestroyprob(prob);
  XPRSfree();
  return 0;
}
```

This solves the very simple optimization problem

$$
\begin{aligned}
&\text{minimize} \\
&\qquad 3x_1 + 5x_2 \\
&\text{subject to} \\
&\qquad 2x_1 + x_2 \geqslant 3 \\
&\qquad 2x_1 + 2x_2 \geqslant 5 \\
&\qquad x_1 + 4x_2 \geqslant 4 \\
&\qquad x_1,\, x_2 \geqslant 0
\end{aligned}
$$

and displays the results.

1.1 Windows

We assume that you have the `cl` compiler installed. The instructions provided here probably work with other compilers (mutatis mutandis) as well, but we tested them only with the `cl` compiler. We also assume that you have setup your console environment for use with that compiler.

Remember that we assume that Xpress was installed into `C:\xpressmp`. Adjust the paths in the commands below if you installed it into a different folder.

When compiling the source file into an object file the only Xpress-specific thing the compiler needs to know is where to find the `xprs.h` header file. The directory that includes that file is specified using the `/I` option and the object file is created like so:

```
cl /IC:\xpressmp\include /c /Foapp.obj app.c
```

When compiling a C++ application you should ask the compiler to correctly define the `__cplusplus` macro since the Xpress C++ header files use this macro. Since the Xpress C++ API requires C++17 it also makes sense to explicitly ask the compiler to support that standard:

```
cl /IC:\xpressmp\include /Zc:__cplusplus /std:c++17 /c /Foapp.obj app.cpp
```

Once object files are created, they have to be linked into an executable. For this the compiler (or linker) needs to know what symbols the Xpress library provides. To this end, the `xprs.lib` file is added to the link step:

```
cl /Feapp.exe app.obj C:\xpressmp\lib\xprs.lib
```

In order to run the application, you need to make sure that the Xpress libraries can be found at runtime.

One way to do this is to copy `xprs.dll` and `xprl.dll` from `C:\xpressmp\bin` to the same place as your application (`exe` file).

Another option is to modify the `PATH` environment variable to contain the `C:\xpressmp\bin` directory.

Once you made the libraries available, you can run the application from the console as

```
app.exe
```

1.1.1 Creating an application with Visual Studio

1. Create a new project of type "C++ Console Application" (choose this even if you want to create a C program).
2. Make sure your appliction is configured for platform "x64" (and not "x86"). The configured platform is displayed in the top bar. If necessary, you can change it there, or from the menu bar go to "Project -> Properties" and modify the "Platform".
3. Replace the default template source code file by the file above or (if your project is empty) add the above source code file to your project.
4. Go to "Project -> Properties".
5. In the dialog perform the following changes (if you installed Xpress into another folder than `C:\xpressmp` then adjust the information accordingly):
   - In "Configuration Properties -> C/C++ -> General" add `C:\xpressmp\include` to "Additional Include Directories".
   - In "Configuration Properties -> Linker -> General" add `C:\xpressmp\lib` to "Additional Library Directories".
   - In "Configuration Properties -> Linker -> Input" add `xprs.lib` to "Additional Dependencies".
   - If the source code uses the Xpress C++ header files then in "Configuration Properties -> C/C++ -> Command line" add `/Zc:__cplusplus` to the "Additional Options" section.
6. At this point your configuration is complete for building your project. You can not yet run it from Visual Studio. To run your project you must make the Xpress DLLs available to the application. One way to do that is copying them to your project directory as follows:
   - Select "Project -> Add Existing Item..."
   - Select `C:\xpressmp\bin\xprs.dll`.
   - Then find the `xprs.dll` entry in the project explorer, right click on it and select "Properties ...".
   - In the dialog change the "General -> Item Type" property to "Copy file" and click "Ok".
   - Do the same with `C:\xpressmp\bin\xprl.dll`.
7. Unless you made your license globally available in some way, you have to set the `XPAUTH_PATH` environment variable so that Xpress can find your license. To this end, select your application in the solution explorer. Then go to "Project -> Properties -> Configuration Properties -> Debugging" and add the appropriate setting of `XPAUTH_PATH` to "Environment". This would be something like `XPAUTH_PATH=/path/to/xpauth.xpr`.
8. Now you can run your project by selecting "Debug -> Start Debugging" or "Debug -> Start Without Debugging" from the menu bar.

1.2 Unix

When compiling the source file into an object file the only Xpress-specific thing the compiler needs to know is where to find the `xprs.h` header file. The directory that includes that file is specified using the `-I` option. In a console issue (remember that we assume you installed Xpress into `/xpressmp`, adjust the path if you installed into a different folder):

```
cc -I/xpressmp/include -c -o app.o app.c
```

If you are using the Xpress C++ API then you should use a C++ compiler instead and tell the compiler to support at least C++17:

```
c++ -std=c++17 -I/xpressmp/include -c -o app.o app.cpp
```

Once object files are created, they have to be linked into an executable. For this the compiler (or linker) needs to know where the Xpress library is located (specified via the `-L` option) and that is has to link with that library (specified via the `-l` option):

```
cc -L/xpressmp/lib -o app app.o -lxprs -lm
```

If you are using the Xpress C++ API then you also need to link `libxprsccx.so` and should use `c++` for linking:

```
c++ -L/xpressmp/lib -o app app.o -lxprs -lm
```

In order to run the application, you need to make sure that the runtime linker can find the Xpress libraries. There are different ways to do that:

1.2.1 Configuring the environment to find Xpress libraries

The runtime linker looks at environment variables to find libraries required by the application. The environment variable has different names on different operation systems:

- On Linux the environment variable is called `LD_LIBRARY_PATH`. To run the application in a shell do

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/xpressmp/lib
./app
```

- On MacOS the environment variable is called `DYLD_LIBRARY_PATH`. To run the application in a shell do

```
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/xpressmp/lib
./app
```

Note that in almost all shells you can also specify environment variables at the time you execute a command. That allows setting them without ”polluting” the global namespace. For example, on Linux using bash you can do:

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/xpressmp/lib ./app
```

1.2.2 Hard coding the library location into the binary

If you know the place at which Xpress libraries are located at runtime then you can hard-code that

location into the application binary. With that the binary will run without any additional environment settings.

In order to hard-code the library location into the binary, add the following flags to the linker command (we assume you link with gcc or clang):

- Linux: `-Wl,--enable-new-dtags,-rpath,/xpressmp/lib`
- MacOS: `-Wl,-rpath,/xpressmp/lib`

If you added the above flags to the linker command, for example on Linux,

```
cc -L/xpressmp/lib -Wl,--enable-new-dtags,-rpath,/xpressmp/lib -o app app.o -lxprs -lm
```

then you can execute the application as

```
./app
```

If your application will be in the same runtime location as the libraries (in the same folder), then instead of `/xpressmp/lib` you can specify `$ORIGIN` on Linux (note that the '$' may have to be quoted) and `@executable_path` on MacOS. Either of the two will make the runtime linker search the folder containing the application for libraries.

1.2.3 Creating an application with Xcode

1. Create new project using the "Command Line Tool" template, found on the "macOS" tab.
2. Fill in the necessary fields, selecting "C++"" as the language.
3. Replace the default template source code file by the file above.
4. Click your project in the left-hand pane, open the "Build Settings" tab, and adjust the following:
   - In the "Search Paths" section, add `/xpressmp/include` to "Header Search Paths".
   - In the "Search Paths" section, add `/xpressmp/lib` to "Library Search Paths".
   - In the "Linking - General" section, add `/xpressmp/lib` to "Runtime Search Paths".
5. Now open the "Build Phases" tab and do the following:
   - Find the "Link Binary With Libraries" section.
   - Click the plus button, and in the dialog, click "Add Other" and select "Add Files".
   - Browse to `/xpressmp/lib` and select `libxprs.dylib`.
6. Unless you made your license globally available in some way, you have to set the `XPAUTH_PATH` environment variable so that Xpress can find your license:
   - Hold the "Alt" key and click the run button in the toolbar.
   - Select "Run" in the left-hand pane.
   - Select the "Arguments" tab and add the appropriate setting of `XPAUTH_PATH` to "Environment Variables". This would be something like `XPAUTH_PATH=/path/to/xpauth.xpr`.
7. Now you can run your project by clicking the run button in the toolbar.

1.3 Example makefiles

Xpress also ships with examples and makefiles to build them. These files are located in `/xpressmp/examples/solver/optimizer/c` and `/xpressmp/examples/solver/nonlinear/c`.

The makefiles assume that the `XPRESSDIR` variable is set to the installation directory of Xpress. `XPRESSDIR` can be set either as environment variable or as makefile variable.

In order to build an example using the provided makefiles you can either build the example in the installation or copy it and build it in a different place. We illustrate this for the `tsp.c` example on Linux:

- Building the example in the installation:

```
cd /xpressmp/examples/solver/optimizer/c
make XPRESSDIR=/xpressmp tsp
```

- Copying the example and building it in a different place:

```
cp /xpressmp/examples/solver/optimizer/c/tsp.c .
make XPRESSDIR=/xpressmp -f /xpressmp/examples/solver/optimizer/c/makefile tsp
```

On Windows things are almost identical, you just have to use `nmake` instead of `make` and the target has to be `tsp.exe` instead of just `tsp`.

Note that building examples through the provided makefiles also gives an easy way to figure out the required compiler and linker flags: when running the makefiles, `make` shows the commands that are executed. From this output you can read the flags required to compile and link Xpress applications.

1.3.1 Linking with the static Xpress library (Linux)

On Linux the Solver library is also available as static library: `libxprs-static.a`. Linking with the static library reduces your application's dependencies by one. Note that the dependency on `libxprl.so` is still there, though.

The compilation (object file creation) step is the same as described above. The only thing that changes is the link step:

```
cc -L/xpressmp/lib -o app-static app.o -lxprs-static -lxprl -lm -lpthread -ldl
```

The generated binary will not require the `libxprs.so` library at runtime since the relevant parts of this library are now contained in the application binary.

1.4 Create Xpress Solver Applications in Eclipse

In order to build C/C++ applications in Eclipse, you need the CDT project. We assume you have this available in your Eclipse installation and have configured everything properly so that you can build C/C++ applications (you can build a "Hello World" program). Below we only specify the additional things you have to add to your C/C++ project configuration to make the Xpress Solver libraries available.

Remember that we assume that Xpress was installed into `C:\xpressmp` on Windows and `/xpressmp` on other platforms. Adjust the paths in the configurations below if you installed it into a different folder.

1. Select your project in the Project Explorer, select "Project -> Properties"
2. Go to "C/C++ General -> Paths and Symbols"

3. On the "Includes" tab add `/xpressmp/include`
4. On the "Library Paths" tab add `C:\xpressmp\bin` on Windows and `/xpressmp/lib` on other platforms
5. On the "Libraries" tab add `xprs` and `xprl`. On Windows these are `xprs.dll` and `xprl.dll` located in `C:\xpressmp\bin`. On Linux these are `libxprs.so` and `libxprl.so` located in `/xpressmp/lib`. On MacOS these are `libxprs.dylib` and `libxprl.dylib` located in `/xpressmp/lib`. For C++ on Windows you should also add `/Zc:__cplusplus` and `/std:c++17` to the compiler flags. For Unix platforms you should add `–std=c++17` (or a higher C++ standard) to the compilation flags for C++ applications.
6. Click "Apply and Close"
7. Open the run configuration for your project
8. Go to the "Environment" tab
9. Depending on your platform, add an environment variable that points to the location of the Xpress Solver libraries:
   - On Linux add `LD_LIBRARY_PATH` pointing to `/xpressmp/lib`
   - On MacOS add `DYLD_LIBRARY_PATH` pointing to `/xpressmp/lib`
   - On Windows add `PATH` pointing to `/xpressmp/bin`

   Note that you may have to include the original value of the respective variable in your setting.
10. Unless you made your license globally available, add an environment variable `XPAUTH_PATH` that points to your `xpauth.xpr` file.
11. Click "Ok"

CHAPTER 2

Creating Xpress Solver Applications in Java

In this chapter we illustrate how to create a Java application that uses the Xpress Solver. We use a very simple application that only consists of this simple `App.java` file:

```
import static com.dashoptimization.XPRSconstants.PLUSINFINITY;
import static com.dashoptimization.XPRSconstants.NAMES_COLUMN;

import com.dashoptimization.XPRSprob;

/* A very simple Xpress application that solves this optimization problem:
 * <pre>
 *    Minimize
 *      3x1 + 5x2
 *    Subject To
 *      2x1 +  x2 >= 3
 *      2x1 + 2x2 >= 5
 *       x1 + 4x2 >= 4
 *      x1,x2 >= 0
 * </pre>
 * and displays the results.
 * For more detailed examples please see the examples/solver/optimizer/java
 * directory in your Xpress installation.
 */
public class App {
    public static void main(String[] args) {
        try (XPRSprob prob = new XPRSprob("")) {
            prob.loadLp("problem", 2, 3,
                    new byte[]{ 'G', 'G', 'G' },                // rowtype
                    new double[] { 3, 5, 4 },                   // rhs
                    null,
                    new double[] { 3, 5 },                      // obj
                    new int[] {0, 3, 6 },                       // start
                    null,
                    new int[] { 0, 1, 2, 0, 1, 2 },             // rowind
                    new double[] { 2, 2, 1, 1, 2, 4 },          // rowcoef
                    new double[] { 0, 0 },                      // lb
                    new double[] { PLUSINFINITY, PLUSINFINITY } // ub
                    );
            prob.addNames(NAMES_COLUMN, new String[] { "x1", "x2"}, 0, 1);
            prob.optimize();
            double[] x = prob.getSolution();
            System.out.printf("x1 = %f%n", x[0]);
            System.out.printf("x2 = %f%n", x[1]);
        }
    }
}
```

This solves the very simple optimization problem

$$\begin{aligned} &\text{minimize} \\ &\qquad 3x_1 + 5x_2 \\ &\text{subject to} \\ &\qquad 2x_1 + x_2 \geqslant 3 \\ &\qquad 2x_1 + 2x_2 \geqslant 5 \\ &\qquad x_1 + 4x_2 \geqslant 4 \\ &\qquad x_1,\ x_2 \geqslant 0 \end{aligned}$$

and displays the results.

Note that there are much more convenient ways to create models (see the examples shipped with Xpress). Here we focus on how creating the application binaries and not the application content.

2.1 Compiling source files that use the Xpress Solver

In order to compile a Java source file that references the Xpress Solver you have to add the `xprs.jar` file to the classpath. This is done using the `-cp` argument for the Java compiler.

- On Windows:

```
javac -cp C:\xpressmp\lib\xprs.jar App.java
```

- On other platforms:

```
javac -cp /xpressmp/lib/xprs.jar App.java
```

The above assumes that Xpress was installed into `C:\xpressmp` for Windows and into `/xpressmp` for other platforms. Adjust paths accordingly if you installed into another folder.

There are also other ways to set the classpath, for example by setting the `CLASSPATH` environment variables.

Xpress also ships example makefiles in `examples/solver/optimizer/java/makefile` and `examples/solver/nonlinear/java/makefile` that illustrate how to compile Java source files that use the Xpress Solver classes. You can do

```
cd /xpressmp/examples/solver/optimizer/java
make TSP.class
```

or

```
cp /xpressmp/examples/solver/optimizer/java/TSP.java .
make -f /xpressmp/examples/solver/optimizer/java/makefile TSP.class
```

to run these makefiles either in your installation or locally. Note that on Windows you have to replace the path names by Windows path names. Also remember that `/xpressmp` denotes the Xpress installation directory. If you installed into a different directory then adjust the paths accordingly.

2.2 Running an application that uses the Xpress Solver

After creating the class files for the application (see the previous section), the application can be run. We must tell the Java runtime environment where to find the Xpress Solver classes and the Solver libraries. This is done by the `-cp` and `-Djava.library.path` options respectively (assuming `/xpressmp` or `C:\xpressmp` is the Xpress installation folder).

- On Windows:

```
java -cp C:\xpressmp\lib\xprs.jar;. -Djava.library.path=C:\xpressmp\bin App
```

- On other platforms:

```
java -cp /xpressmp/lib/xprs.jar:. -Djava.library.path=/xpressmp/lib App
```

Note that on Windows the classpath separator is ';', while on other platforms it is ':'. Also, on Windows the `java.library.path` points to the `bin` directory, while for other platforms it points to the `lib` directory. Another additional prerequisite for Windows is to set the `PATH` environment variable to contain `C:\xpressmp\bin`. Otherwise the Java runtime environment cannot find dependent Xpress libraries.

2.3 Setting up Eclipse for Xpress

In order to create an application with the above example file in Eclipse follow the steps below. They assume that Xpress was installed into `C:\xpressmp` on Windows and into `/xpressmp` on other platforms. If you installed into a different directory then adjust the paths accordingly.

1. Create a new Java project called "XpressApplication"
2. Select your project and choose "Project -> Properties"
3. In the dialog that opens select "Java Build Path" and go to the "Libraries" tab.
4. On the "Libraries" tab click "Add External JARs..." and find and select the `/xpressmp/lib/xprs.jar` JAR (`C:\lib\xprs.jar` on Windows).
5. Click "Apply and Close"
6. Add the `App.java` file listed above to your project.
7. Create a new run configuration for your project:
   (a) Choose "Run -> Run Configurations ..."
   (b) In the dialog create a new "Java Application" with the following settings
      - Set the "Name" to something reasonable
      - On the "Main" tab select the `App` class as main class.
      - On the "Arguments" tab add `-Djava.library.path=C:\xpressmp\bin` for Windows and `-Djava.library.path=/xpressmp/lib` for other platforms to the "VM Arguments" section.
      - On the "Environment" tab add a new variable `XPAUTH_PATH` that points to your `xpauth.xpr` file.
      - On Windows also create a `PATH` variable with value `C:\xpressmp\bin`.
      - On the "Dependencies" tab add `/xpressmp/lib/xprs.jar` (`C:\lib\xprs.jar` on Windows) as external JAR to the "Classpath Entries" element.
      - Click "Apply".
   (c) Execute the run configuration (run your application) by clicking on "Run".

It is also possible to link the Xpress javadoc with your application in Eclipse. This way you will get documentation if you hover over classes, functions, etc. that are defined in the Xpress JAR. In order to link Xpress javadoc perform the following:

1. Locate `xprs.jar` in the project explorer.

2. Right-click and choose "Properties".
3. Go to "Javadoc location".
4. Check "Javadoc in archive".
5. Set the "Archive path" to `/xpressmp/lib/xprs-javadoc.jar` (`C:\xpressmp/\lib\xprs-javadoc.jar` on Windows).
6. Click "Apply and Close".

If you now hover for example over "XPRSprob" in the source code, you should see a short description of the `XPRSprob` class.

CHAPTER 3

Creating Xpress Solver Applications in C#

In this chapter we illustrate how to create a C# application that uses the Xpress Solver. We use a very simple application that only consists of this simple `App.cs` file:

```
using System;
using Optimizer;

namespace XpressApplication
{
  /// <summary>A very simple Xpress application.</summary>
  /// <remarks>
  /// The application that solves this optimization problem:
  ///  Minimize
  ///    3x1 + 5x2
  ///  Subject To
  ///    2x1 +  x2 >= 3
  ///    2x1 + 2x2 >= 5
  ///     x1 + 4x2 >= 4
  ///     x1,x2 >= 0
  /// and displays the results.
  /// For more detailed examples please see the directory
  /// examples/solver/optimizer/csharp in your Xpress installation.
  /// </remarks>
  class App
  {
    static void Main(string[] args)
    {
      using (XPRSprob prob = new XPRSprob(""))
      {
        prob.LoadLP("problem", 2, 3,
          new char[] { 'G', 'G', 'G' },                          // rowtype
          new double[] { 3, 5, 4 },                              // rhs
          null,
          new double[] { 3, 5 },                                 // obj
          new int[] { 0, 3, 6 },                                 // start
          null,
          new int[] { 0, 1, 2, 0, 1, 2 },                        // rowind
          new double[] { 2, 2, 1, 1, 2, 4 },                     // rowcoef
          new double[] { 0, 0 },                                 // lb
          new double[] { XPRS.PLUSINFINITY, XPRS.PLUSINFINITY } // ub
        );

        prob.AddNames(Optimizer.Namespaces.Column,
          new String[] { "x1", "x2" }, 0, 1);
        prob.Optimize();
        double[] x = prob.GetSolution();
        Console.WriteLine("x1 = {0}", x[0]);
        Console.WriteLine("x2 = {0}", x[1]);
      }
    }
  }
}
```

This solves the very simple optimization problem

$$\begin{aligned} &\text{minimize} \\ &\qquad 3x_1 + 5x_2 \\ &\text{subject to} \\ &\qquad 2x_1 + x_2 \geqslant 3 \\ &\qquad 2x_1 + 2x_2 \geqslant 5 \\ &\qquad x_1 + 4x_2 \geqslant 4 \\ &\qquad x_1,\ x_2 \geqslant 0 \end{aligned}$$

and displays the results.

Note that there are much more convenient ways to create models (see the examples shipped with Xpress). Here we focus on how creating the application binaries and not the application content.

3.1 Creating Xpress applications in Visual Studio

3.1.1 Using NuGet packages

The Xpress Solver library is shipped as a NuGet package. Those packages have to be registered with Visual Studio. To do that, perform the following steps (with or without a project/solution) loaded (we assume that Xpress was installed into `C:\xpressmp`, adjust paths in the below configuration if you installed into a different folder):

1. Select "Tools -> NuGet Package Manager -> Package Manager Settings"
2. Select "NuGet Package Manager -> Package Sources"
3. Click on the "+" icon to add a package with default settings.
4. Modify the default settings:
   - Choose a reasonable name, for example "Xpress x.y.z" where x.y.z is the Xpress version number.
   - Set the source to `C:\xpressmp\lib\nuget`
5. Click "Update" and make sure the newly added package source is checked in the list of package sources.
6. Click "Update" then "Ok" to save the changes.

This is a one-time operation that you have to perform only once (or once for every version of Xpress that you install).

Now to create a C# application that uses the Xpress Solver, follow these steps:

1. Create a new "C# Console Application" project (choosing an appropriate target framework).
2. Select "Project -> Manage NuGet Packages ...
3. From the pull down select the Xpress Solver package you added above
4. Go to the "Browse" tab.
5. Select `FICO.Xpress.XPRSdn` and click "Install"
6. Go to "Project -> Add Existing Item ..."

7. Change the file filter to either ”All Files” or ”Executable Files”.
8. Find the `C:\xpressmp\bin\xprs.dll` library and ”Add” it.
9. In the project explorer right-click the `xprs.dll` item and select ”Properties”.
10. Change the ”Copy to Output Directory” property to either ”Copy always” or ”Copy if newer”.
11. Add the `C:\xpressmp\bin\xprl.dll` in the same way.
12. Copy the above source code over the default main source code file in your newly created project.
13. Now your project is ready for compilation. In order to run it you must also point Xpress to your license path location:
    - Open your project’s property dialog.
    - In ”Debug ->Environment Variables” add a variable `XPAUTH_PATH` that points to the absolute path to your `xpauth.xpr` file.

Appendix A

Contacting FICO

FICO provides clients with support and services for all our products.

FICO Customer Support

FICO Customer Support offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have an active maintenance contract.

The FICO Customer Self-Service Portal (support.fico.com) is a secure web portal that allows users to open, review, and update their support cases; manage their organization's portal users; find solutions to common problems in the FICO Knowledge Base; and view the availability of their cloud applications 24 hours a day, 7 days a week.

You can find support contact information and a link to the FICO Customer Self-Service Portal (online support) on the Product Support home page (www.fico.com/en/product-support).

Please include *'Xpress'* in the subject line of your support queries.

Documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide.

If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com. Please include your contact information (name, company, email address, and optionally, your phone number) so we may reach you if we have questions.

FICO Learning

FICO Learning is the principal provider of product training for our clients and partners. FICO Learning offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support.

For additional information, visit the FICO Learning home page at www.fico.com/en/product-training or email producteducation@fico.com.

Sales and maintenance

If you need information on other Xpress Optimization products, or you need to discuss maintenance contracts or other sales-related items, contact FICO by:

- Phone: +1 (408) 535-1500 or +44 207 940 8718
- Web: www.fico.com/optimization and use the available contact forms

About FICO

FICO (NYSE:FICO) is a leading analytics software company, helping businesses in 90+ countries make better decisions that drive higher levels of growth, profitability, and customer satisfaction. Learn more at www.fico.com or contact us at www.fico.com/en/contact-us.