Reference manual

FICO

FICO® Xpress Optimization

mosjvm Module for Mosel

Reference Manual

Release 8.3

Last update 17 July, 2017

FICO® Xpress Optimization

Deliverable Version: A

Last Revised: 17 July, 2017

Version 8.3

Contents

CHAPTER 1

Introduction

The *mosjvm* module allows you to create a Java virtual machine within the same process running Mosel, and call static Java methods from within a Mosel model. For example:

```
javaReturnString :=
  jvmcallstr('com.fico.testpackage.TestClass.myTestFunction', functionArg1, functionArg2,
             functionArg3)

writeln("2+5=",
        jvmcallint('com.fico.testpackage.SimpleMathOperations.addTwoValues', 2, 5))
```

In this way, you can interact with pre-existing Java libraries from your Mosel model. However, please note that there are some drawbacks to instantiating a Java virtual machine in the same process as the Mosel virtual machine, and developers are advised to consider other approaches (for example: deploying your Java classes to a Tomcat webserver and interacting with them using standard webservice requests) before building a solution based upon *mosjvm*.

The developer is advised to read the later chapter which goes into some detail about the limitations of this module. But the most major limitations to be aware of are:

- This module is only available for Windows and Linux.
- This module can not be used within the FICO Analytic Cloud.
- Java 8 must be installed on the machine running the model.

CHAPTER 2

Usage

2.1 Calling Java

Add `uses 'mosjvm'` to the top of your model then you can call any static Java method by passing its full class and method name to one fo the `jvmcall<type>` functions, where `<type>` represents the return type of the function, for example:

```
model myModel
  uses 'mosjvm'

  writeln('Integer return value=', jvmcallint('com.fico.MyClassName.myIntMethod'))
  writeln('String return value=', jvmcallstr('com.fico.MyClassName.myStringMethod'))
  writeln('Real return value with two parameters=',
          jvmcallreal('com.fico.MyClassName.myRealMethod', 5.7, true))
end-model
```

You can call *jvmcallvoid* if your Java method doesn't return anything. Any number of method arguments may be specified, from the following types:

- boolean (translates to Java type: boolean)
- integer (translates to Java type: int)
- real (translates to Java type: double)
- string (translates to Java type: java.lang.String)
- text (translates to Java type: java.lang.String)
- array of boolean (translates to Java type: boolean[])
- array of integer (translates to Java type: int[])
- array of real (translates to Java type: double[])
- array of string (translates to Java type: java.lang.String[])
- array of text (translates to Java type: java.lang.String[])

Where an argument is an array, it must be indexed by a Mosel range starting from 0, *e.g.*:

```
model myModelAr
  uses 'mosjvm'
  declarations
    MyArray: array(range) of string
  end-declarations
```

```
  MyArray(0) := "zero"
  MyArray(1) := "one"
  MyArray(2) := "two"

  jvmcallvoid("com.fico.MyClassName.myMethodTakingArrayOfString", MyArray)
end-model
```

2.2 Starting Java

The first time a `jvmcall<type>` function is called from a model in a Mosel process, *mosjvm* will load and initialize a Java virtual machine. This Java virtual machine will then be used by all subsequent `jvmcall<type>` functions called by models in this process—if your Java functions modify any static fields, those modified values will be read by subsequent calls, possibly from different models. If you are using *mosjvm* from multiple submodels within the same Mosel instance, your Java code will need to be threadsafe.

By default, *mosjvm* will look for the Java Runtime Environment in the following locations:

- The folder specified by the `MOSJVM_JAVA_HOME` environment variable, if set
- The folder specified by the `JAVA_HOME` environment variable, if set
- Relative to the location of a `java` executable found in a folder on the `PATH` environment variable
- In some common install locations

To force *mosjvm* to use a specific installation of Java, set the environment variable `MOSJVM_JAVA_HOME` before starting Mosel.

2.3 Classpath

By default, *mosjvm* will look for your Java classes using a classpath determined as follows:

- The classpath specified by the `MOSJVM_CLASSPATH` environment variable, if set
- Otherwise, the classpath specified by the `CLASSPATH` environment variable, if set
- Otherwise, the current working directory of the Mosel process

2.4 Exception handling

When an error occurs within a Java method, the Java Virtual Machine will throw an *exception*. From Mosel, you can check whether your last call to Java threw an exception by looking at the `jvmcallstatus` parameter; a non-zero value indicates an exception. When `jvmcallstatus` is non-zero, the `jvmexceptiontype` parameter will hold the class name of the exception (*e.g.* `java.io.FileNotFoundException`) and the `jvmexceptionmsg` parameter will contain the exception's text. For example:

```
model myModelExc
  uses 'mosjvm'

  jvmcallvoid('com.fico.MyClassName.parseMyInputFile', 'inputfile.dat')
  if getparam('jvmcallstatus')<>0 then
```

```
    setioerr('Encountered Java error of type: '+ getparam('jvmexceptiontype')+", message: "+getparam('jvmexceptio
  end-if
end-model
```

If you set the parameter `jvmdebug` to 'true', the stack trace of any exceptions thrown by your calls to Java will be written to the Mosel model's error stream.

CHAPTER 3

Limitations

3.1 No complex return types

Java methods can only return values of basic types back to Mosel: boolean, integer, real, string and text. If you need to return a large data-set then your options would be to format it in some standard format (*e.g.* CSV, XML, JSON), and return it as 'text' then use Mosel's standard modules to parse it, or to write it to a file that can be read later by the Mosel model.

3.2 Arguments passed by value

Method arguments are input-only—so when you make changes to an array from within a Java function, this does not affect the array in the Mosel model. Similarly when you pass a Mosel text object, it becomes a Java String and cannot be modified from Java.

3.3 Standard output streams

If you send output to the Java stream `System.out` within a Java method called through *mosjvm*, the output will be sent to the Mosel model's standard output stream. However, if the Java virtual machine generates output from a different thread, this will not be captured by any Mosel stream.

Similarly, output you send to `System.err` will go to the the Mosel model's error stream. If another thread generates output to `System.err`, it will be sent to the Mosel instance's error stream.

3.4 Working directories

The initial working directory of the Java virtual machine will be the working directory of the Mosel process; this may be different from the working directories of the Mosel models (controlled by the `workdir` parameter). When passing filenames between the Mosel and Java environments, it is recommended to always use absolute paths.

3.5 Static methods only

mosjvm can only be used to call static methods. It is not possible to call instance methods or reference a Java object directly from a Mosel model.

3.6 JVM shut down

In normal usage, once the Java virtual machine has started up, it will not be shut down until the Mosel process exits.

3.7 Incompatible with `com.dashoptimization.XPRM`

If you are calling Mosel using the Mosel Java Library classes in the `com.dashoptimization` package, you cannot use the *mosjvm* module.

3.8 Incompatible with other JVMs

The *mosjvm* module can only interact with a Java virtual machine that it itself has started. If there is already a Java virtual machine running in the Mosel process, attempting to use the *mosjvm* module will result in an error.

3.9 Supported Java version

The *mosjvm* module is only supported with Oracle or OpenJDK Java 8. It has not been tested with other versions of Java.

3.10 Window and Linux only

The *mosjvm* module is only available with Windows and Linux versions of Xpress. It is not available for other platforms.

3.11 ThreadLocal values not supported

Every Java call made using the *mosjvm* module occurs in a separate Java thread that is terminated after the Java method completes. This means that you cannot use ThreadLocal objects to share values between multiple `jvmcall<type>` invocations from the same model. If you are running multiple models within the Mosel instance and need to store some per-model data in the Java runtime, it's recommended you store them in a static `ConcurrentHashMap` field, using the value of the Mosel `modelname` or `modelnumber` parameter as a key, and make an extra call at the end of your model to clean up any stored data for this model.

3.12 Not supported in FICO Analytic Cloud

The use of *mosjvm* is not supported by applications running within the FICO Analytic Cloud.

3.13 `System.exit` may not be called

Calling `System.exit` from the JVM started by *mosjvm* will not work; an exception will be thrown.

CHAPTER 4

Use in Mosel Restricted Mode

When Mosel is run in restricted mode, the JVM will be started with a security manager that will try to match the Mosel security model. For example, if the Mosel *WDOnly* restriction is present, you will be prevented from reading from or writing to any files outside of the Mosel instance's working directory.

The JVM will always be able to read from the Java installation directory, and any configured Java extension directories, regardless of the Mosel restrictions level.

When the *NoExec* restriction is present, the use of Java is subject to several additional restrictions:

- Other processes may not be started from Java.
- Native libraries may not be loaded by the Java Virtual Machine, except for those installed within the Java installation directory.
- The JVM's classpath shall only be read from the environment variable `MOSJVM_CLASSPATH`.
- The environment variable `MOSJVM_ALLOW` must contain a whitespace-separated list of the classes that can be called using a `jvmcall<type>` function.

When any Mosel security restriction is present (*i.e.* `MOSEL_RESTR` is non-zero), the JVM will additionally be restricted as follows:

- The JVM may not open any GUI windows.
- `System.setSecurityManager` may not be called.
- Direct access to file descriptors is disabled.
- Access to the printing API is disabled.
- Access to the audio API is disabled.
- Access to the reflection API is disabled.
- Access to the preferences backing store is disabled.
- Code may not add additional classes with package names starting `java.` or `sun.`
- Java code may not set cookie handlers, proxy selectors, response caches or change the default authenticator.
- Java code may not set SSL hostname verifiers or change the default SSL context.
- Creation of sockets is disabled.
- Access to private credentials is disabled.

- Access to Kerberos delegation is disabled.

The above list is not intended to be exhaustive; the permissions required to perform Java operations are sometimes non-obvious. When building a solution that will use *mosjvm* in an environment where Mosel security restrictions are in place, it is recommended that the developer performs testing with the required Mosel security restrictions as early as possible.

CHAPTER 5

Examples

5.1 Passing values to a Java method

This example demonstrates calling a Java method and returning a value to Mosel.

```
model MosJvmExample1
  uses "mosjvm"

  parameters
    SRC_VALUE=5
  end-parameters

  writeln(SRC_VALUE, "*2=",
    jvmcallint("com.fico.examples.MathOperations.multiply", SRC_VALUE, 2))
  if getparam("jvmcallstatus")<>0 then
    setmatherr("Java exception: "+ getparam("jvmexceptiontype"))
  end-if

end-model
```

Where the `MathOperations` class is defined as follows:

```
package com.fico.examples;

public class MathOperations {
  public static int multiply(int v1,int v2) {
    return v1*v2;
  }
}
```

5.2 Calling a void Java method

This example demonstrates calling a Java method and displaying a value to the Mosel output stream.

```
model MosJvmExample2
  uses "mosjvm"

  parameters
    SRC_VALUE=5
  end-parameters

  jvmcallvoid("com.fico.examples.MathOperations.multiplyAndDisplay", SRC_VALUE, 2)
  if getparam("jvmcallstatus")<>0 then
    setmatherr("Java exception: "+ getparam("jvmexceptiontype"))
  end-if
```

```
end-model
```

Where the `MathOperations` class is defined as follows:

```
package com.fico.examples;

public class MathOperations {
  public static void multiplyAndDisplay(int v1,int v2) {
    System.out.println("The result of "+v1+"*"+v2+" is "+(v1*v2));
  }
}
```

5.3 Passing an array to a Java method

This example demonstrates passing an array to a Java method and returning a value to Mosel.

```
model MosJvmExample3
  uses "mosjvm"

  declarations
    MyData: array(range) of real
    result: real
  end-declarations

  ! Create some data
  forall(x in 0..5000) MyData(x) := x*5

  ! Calculate sum
  result := jvmcallreal("com.fico.examples.MathOperations.sumOfArray", MyData)
  if getparam("jvmcallstatus")<>0 then
    setmatherr("Java exception: "+ getparam("jvmexceptiontype"))
  end-if

  ! Print result
  writeln("Sum of values = ", result)

end-model
```

Where the `MathOperations` class is defined as follows:

```
package com.fico.examples;

public class MathOperations {
  public static double sumOfArray(double[] values) {
    double tot = 0;
    for (double v : values) {
      tot += v;
    }
    return tot;
  }
}
```

CHAPTER 6

Procedures and functions

jvmcallint

Purpose

Call a public static int function in a Java class.

Synopsis

```
function jvmcallint(qualifiedmethodname:string) :  integer
function jvmcallint(qualifiedmethodname:string, ...)  :  integer
```

Arguments

`qualifiedmethodname`	The name of the method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`...`	Following the method name, specify zero or more values to pass as arguments to the Java method.

Further information

1. *mosjvm* will look in the class for a method with the given argument types that is declared as returning an `int`. If such a method cannot be found, the model will terminate with a runtime error.
2. The types of value that may be passed as method arguments are discussed earlier in this guide.
3. If *mosjvm* has not yet loaded the Java Virtual Machine into the Mosel process, it will be loaded by this function. If Java cannot be found or an error occurs loading it, the model will terminate with a runtime error.

Related topics

`jvmcallstatus`, `jvmdebug`

jvmcallreal

Purpose

Call a public static double function in a Java class.

Synopsis

```
function jvmcallreal(qualifiedmethodname:string) :  real
function jvmcallreal(qualifiedmethodname:string, ...)  :  real
```

Arguments

`qualifiedmethodname`	The name of the method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`...`	Following the method name, specify zero or more values to pass as arguments to the Java method.

Further information

1. *mosjvm* will look in the class for a method with the given argument types that is declared as returning a `double`. If such a method cannot be found, the model will terminate with a runtime error.
2. The types of value that may be passed as method arguments are discussed earlier in this guide.
3. If *mosjvm* has not yet loaded the Java Virtual Machine into the Mosel process, it will be loaded by this function. If Java cannot be found or an error occurs loading it, the model will terminate with a runtime error.

Related topics

`jvmcallstatus`, `jvmdebug`

jvmcallstr

Purpose

Call a public static String function in a Java class.

Synopsis

```
function jvmcallstr(qualifiedmethodname:string) :  string
function jvmcallstr(qualifiedmethodname:string, ...)  :  string
```

Arguments

`qualifiedmethodname`	The name of the method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`...`	Following the method name, specify zero or more values to pass as arguments to the Java method.

Further information

1. *mosjvm* will look in the class for a method with the given argument types that is declared as returning a `java.lang.String`. If such a method cannot be found, the model will terminate with a runtime error.
2. The types of value that may be passed as method arguments are discussed earlier in this guide.
3. If *mosjvm* has not yet loaded the Java Virtual Machine into the Mosel process, it will be loaded by this function. If Java cannot be found or an error occurs loading it, the model will terminate with a runtime error.

Related topics

`jvmcallstatus`, `jvmdebug`, `jvmcalltext`

jvmcalltext

Purpose

Call a public static String function in a Java class, returning the value into Mosel as a `text` value.

Synopsis

```
function jvmcalltext(qualifiedmethodname:string) :  text
function jvmcalltext(qualifiedmethodname:string, ...)  :  text
```

Arguments

`qualifiedmethodname`	The name of the method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`...`	Following the method name, specify zero or more values to pass as arguments to the Java method.

Further information

1. *mosjvm* will look in the class for a method with the given argument types that is declared as returning a `java.lang.String`. If such a method cannot be found, the model will terminate with a runtime error.
2. The types of value that may be passed as method arguments are discussed earlier in this guide.
3. If *mosjvm* has not yet loaded the Java Virtual Machine into the Mosel process, it will be loaded by this function. If Java cannot be found or an error occurs loading it, the model will terminate with a runtime error.

Related topics

`jvmcallstatus`, `jvmdebug`, `jvmcallstr`

jvmcallvoid

Purpose

Call a public static void function in a Java class.

Synopsis

```
procedure jvmcallvoid(qualifiedmethodname:string)
procedure jvmcallvoid(qualifiedmethodname:string, ...)
```

Arguments

`qualifiedmethodname`	The name of the method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`...`	Following the method name, specify zero or more values to pass as arguments to the Java method.

Further information

1. *mosjvm* will look in the class for a method with the given argument types that is declared as `void`. If such a method cannot be found, the model will terminate with a runtime error.
2. The types of value that may be passed as method arguments are discussed earlier in this guide.
3. If *mosjvm* has not yet loaded the Java Virtual Machine into the Mosel process, it will be loaded by this procedure. If Java cannot be found or an error occurs loading it, the model will terminate with a runtime error.

Related topics

`jvmcallstatus`, `jvmdebug`

CHAPTER 7

Parameters

Via the `getparam` function and the `setparam` procedure it is possible to access the following control parameters of module *mosjvm* :

jvmcallstatus

Description Will be nonzero if the last call to a Java method threw an exception, zero otherwise.

Type Integer, read only

jvmexceptiontype

Description If the last call to a Java method threw an exception, this parameter will hold the exception class name (*e.g.* `java.io.FileNotFoundexception`). Otherwise, it will be an empty string.

Type String, read only

jvmexceptionmsg

Description If the last call to a Java method threw an exception, this parameter will hold the exception message string (*i.e.* result of calling `Exception.getMessage()`). Otherwise, it will be an empty string.

Type String, read only

jvmloadverbose

Description Set to 'true' to activate some additional logging of how *mosjvm* looks for and finds your installation of Java, to the model's error stream.

Type Boolean, read/write

Default value `false`

Note This can give useful information if it is not clear which Java *mosjvm* is loading, or if it cannot find your installation of Java.

Note This parameter will only have effect if set before the JVM is loaded into the Mosel process.

jvmdebug

Description When set to 'true', if an exception is thrown by a Java method called by *mosjvm*, the exception stack trace will be output to the Mosel error stream.

Type Boolean, read/write

Default value `false`

Note This can be useful when debugging Java exceptions that are thrown by methods you call from Mosel.

jvmxms

Description Allows you to set the initial amount of memory used by the JVM's heap. For more details, please consult the Java documentation for the command-line flag `-Xms`

Type String, read/write

Default value `64m`

Note This parameter will only have effect if set before the JVM is loaded into the Mosel process.

jvmxmx

Description Allows you to set the maximum amount of memory available to the JVM's heap. For more details, please consult the Java documentation for the command-line flag `-Xmx`

Type String, read/write

Default value `512m`

Note This parameter will only have effect if set before the JVM is loaded into the Mosel process.

APPENDIX A

Contacting FICO

FICO provides clients with support and services for all our products. Refer to the following sections for more information.

Product support

FICO 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 purchased a FICO product and have an active support or maintenance contract. You can find support contact information on the Product Support home page (www.fico.com/support).

On the Product Support home page, you can also register for credentials to log on to FICO Online Support, our web-based support tool to access Product Support 24x7 from anywhere in the world. Using FICO Online Support, you can enter cases online, track them through resolution, find articles in the FICO Knowledge Base, and query known issues.

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

Product education

FICO Product Education is the principal provider of product training for our clients and partners. Product Education 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 Product Education homepage at www.fico.com/en/product-training or email producteducation@fico.com.

Product 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.

Sales and maintenance

USA, CANADA AND ALL AMERICAS

Email: XpressSalesUS@fico.com

WORLDWIDE

Email: XpressSalesUK@fico.com

Tel: +44 207 940 8718
Fax: +44 870 420 3601

Xpress Optimization, FICO
FICO House
International Square
Starley Way
Birmingham B37 7GN
UK

Related services

Strategy Consulting: Included in your contract with FICO may be a specified amount of consulting time to assist you in using FICO Optimization Modeler to meet your business needs. Additional consulting time can be arranged by contract.

Conferences and Seminars: FICO offers conferences and seminars on our products and services. For announcements concerning these events, go to www.fico.com or contact your FICO account representative.

About FICO

FICO (NYSE:FICO) delivers superior predictive analytics solutions that drive smarter decisions. The company's groundbreaking use of mathematics to predict consumer behavior has transformed entire industries and revolutionized the way risk is managed and products are marketed. FICO's innovative solutions include the FICO® Score—the standard measure of consumer credit risk in the United States—along with industry-leading solutions for managing credit accounts, identifying and minimizing the impact of fraud, and customizing consumer offers with pinpoint accuracy. Most of the world's top banks, as well as leading insurers, retailers, pharmaceutical companies, and government agencies, rely on FICO solutions to accelerate growth, control risk, boost profits, and meet regulatory and competitive demands. FICO also helps millions of individuals manage their personal credit health through www.myfico.com. Learn more at www.fico.com. FICO: Make every decision count™.

Index