mosjvm Module for Mosel

9.2.0

REFERENCE MANUAL

FICO® Xpress Optimization

FICO

©2017–2026 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: 16 October, 2023

Contents

1 Introduction 1

2 Usage 2

2.1 Calling Java . . . 2
2.2 Starting Java . . . 3
2.3 Classpath . . . 3
2.4 Exception handling . . . 4
2.5 Method Signatures . . . 4

3 Limitations 5

3.1 Primitive value types . . . 5
3.2 Limited complex return types . . . 5
3.3 Arguments passed by value . . . 5
3.4 Standard output streams . . . 5
3.5 Working directories . . . 6
3.6 JVM shut down . . . 6
3.7 Incompatible with com.dashoptimization.XPRM . . . 6
3.8 Incompatible with other JVMs . . . 6
3.9 Supported Java versions . . . 6
3.10 Not supported on Solaris . . . 6
3.11 System.exit may not be called . . . 6

4 Use in Mosel Restricted Mode 7

5 Examples 9

5.1 Passing values to a Java method . . . 9
5.2 Calling a void Java method . . . 9
5.3 Passing an array to a Java method . . . 10
5.4 Calling a Java method that returns an Object . . . 10
5.5 Creating and calling methods on Java objects . . . 11

6 Types 12

6.1 The jvmobject type . . . 12
6.1.1 Attributes . . . 13

7 Procedures and functions 14

getboolvalue . . . 15
getbytevalue . . . 16
getcharvalue . . . 17
getclass . . . 18

getfloatvalue . . . 19
getintvalue . . . 20
getisnull . . . 21
getlongvalue . . . 22
getrealvalue . . . 23
getshortvalue . . . 24
getstrvalue . . . 25
gettextvalue . . . 26
jvmattachthread . . . 27
jvmcallint . . . 28
jvmcallobj . . . 29
jvmcallreal . . . 30
jvmcallstr . . . 31
jvmcalltext . . . 32
jvmcallvoid . . . 33
jvmdetachthread . . . 34
jvmgetexceptionclass . . . 35
jvmgetexceptionmsg . . . 36
jvmnewobj . . . 37
jvmsetcreationargs . . . 38
jvmstatus . . . 39
setnull . . . 40
setvalue . . . 41

8 Parameters 42

jvmabortonexception . . . 42
jvmcontainermemoryoptions . . . 42
jvmclasspath . . . 43
jvmdebug . . . 43
jvminterrupt . . . 43
jvmisloaded . . . 43
jvmloadverbose . . . 44
jvmxms . . . 44
jvmxmx . . . 44

9 Deprecated Functionality 45

9.1 Deprecated Parameters . . . 45
jvmcallstatus . . . 45
jvmexceptiontype . . . 45
jvmexceptionmsg . . . 45

Appendix 46

A Contacting FICO 46

FICO Customer Support . . . 46
Documentation . . . 46
FICO Learning . . . 47
Sales and maintenance . . . 47
About FICO . . . 47

Index 48

CHAPTER 1

Introduction

The *mosjvm* module allows you to create a Java virtual machine within the same process running Mosel, and call 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:

- Java 8, 11 or 17 must be installed on the machine running the model (Java 8 is not supported on Mac ARM). Other versions of Java are not supported at this time. The module is tested on Amazon Corretto.
- This module can only run compiled Java .class files (either directly as files or built into a .jar). Java source code must be compiled by other tools.

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))
  writeln('Any object return value=', jvmcallobj('com.fico.MyClassName.myObjMethod'))
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)
- jvmobject
- 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[])
- array of jvmobject (translates to Java type: java.lang.Object[])

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
```

You can call instance methods on a Java object by passing the object reference as the first parameter and method name as second, *e.g.*:

```
model myModel
  uses 'mosjvm'

  ! Call static method to get object
  sysprops := jvmcallobj('java.lang.System.getProperties')
  ! Call non-static method on that object
  writeln('java.version property value = ',jvmcallstr(sysprops, 'getProperty', 'java.version'))
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 `jvmclasspath` parameter, if set
- Otherwise, 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 calling the `jvmstatus` function; a return value of `false` indicates an exception. When `jvmstatus` is false, the `jvmgetexceptionclass` function will return the class name of the exception (*e.g.* `java.io.FileNotFoundException`) and the `jvmgetexceptionmsg` function will return the exception's message. For example:

```
model myModelExc
  uses 'mosjvm'

  jvmcallvoid('com.fico.MyClassName.parseMyInputFile', 'inputfile.dat')
  if jvmstatus=false then
    setioerr('Encountered Java error of type: '+jvmgetexceptionclass+", message: "+jvmgetexceptionmsg)
  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.

If you set the parameter `jvmabortonexception` to 'true', the Mosel model will immediately terminate with a runtime error if any Java method throws an exception.

2.5 Method Signatures

When you invoke any of the `jvmcall` functions, *mosjvm* will first look for a method with the types exactly matching those passed in. For example, if you pass two integers, *mosjvm* will look for a function signature that takes two int values. If passed an argument of type `jvmobject`, *mosjvm* will look for a method accepting values of the type of the object reference stored in the jvmobject.

If no match is found, *mosjvm* will search the class for any method with the correct name that could accept the parameters passed. If multiple matching methods are found, the one chosen is arbitrary, for example if the arguments are a jvmobject containing a java.lang.Integer and your class defines these methods:

```
public static void myMethod(Object o);
public static void myMethod(Number o);
```

then it is not possible to predict which method *mosjvm* will call.

CHAPTER 3

Limitations

3.1 Primitive value types

Mosel can only directly understand a subset of the primitive Java data-types (boolean, int, double) and their direct Mosel equivalents (boolean, integer, real). When passing other primitive values, they must be wrapped in a `jvmobject`. For example, to pass a timestamp to the `java.util.Date` constructor as a long:

```
declarations
  ts, dt: jvmobject
end-declarations
ts.longvalue := 1648490312000.0
dt := jvmnewobj('java.util.Date',ts)
```

Similarly, to call a method that returns an unsupported primtive type, you can use `jvmcallobj` and then extract the primitive value from the `jvmobject`; for example, the method `java.lang.System.currentTimeMillis()` returns a long, and can be called as follows:

```
declarations
  ticks: jvmobject
end-declarations
ticks := jvmcallobj('java.lang.System.currentTimeMillis')
writeln('currentTimeMillis: ', ticks.longvalue)
```

3.2 Limited complex return types

Mosel can only directly understand the primitive Java data-types and strings. Although your method can return any Java class into a 'jvmobject' variable, the main operations you can do with the object are turning it into the corresponding basic Mosel types or using it in calls to other Java methods.

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.

3.3 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.

3.4 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.

3.5 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.

3.6 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 versions

The *mosjvm* module is only supported with Java 8, 11 and 17. Only the Oracle, OpenJDK and Amazon Corretto distributions of Java are supported. The *mosjvm* module is not supported with other versions or distributions of Java.

3.10 Not supported on Solaris

The *mosjvm* module is not available in the Sun Solaris versions of Xpress.

3.11 `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. If you set a classpath using the `jvmclasspath` parameter, this must contain only jars and directories that are readable under any active Mosel security restrictions (e.g. in the work directory or a directory specified in `MOSEL_ROPATH` or `MOSEL_RWPATH` if `WDOnly` restriction is active).

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` or the parameter `jvmclasspath`.
- The environment variable `MOSJVM_ALLOW` must contain a whitespace-separated list of the classes that can be called using a `jvmcall<type>` function, or '*' if you wish to allow any class to be called.
- Arbitrary JVM arguments may not be specified using `jvmsetcreationargs`.

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.
- When used from a DMP component, environment variables may not be accessed

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.

When using Java 17, a warning will be output to the process error stream when you use `mosjvm` while Mosel restrictions are active:

```
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by com.fico.xpress.mosjvm.MosJvmEnv (file:/C:/xpressmp
WARNING: Please consider reporting this to the maintainers of com.fico.xpress.mosjvm.MosJvmEnv
WARNING: System::setSecurityManager will be removed in a future release
```

This is output by the JVM and cannot be supressed. While it can be ignored, it serves as a warning that using `mosjvm` with Mosel restrictions active will not be possible in later versions of Java (18+).

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 jvmstatus=false then
    setmatherr("Java exception: "+ jvmgetexceptionclass)
  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 jvmstatus=false then
    setmatherr("Java exception: "+jvmgetexceptionclass)
  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 jvmstatus=false then
    setmatherr("Java exception: "+jvmgetexceptionclass)
  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;
  }
}
```

5.4 Calling a Java method that returns an Object

This example demonstrates calling a Java method that returns an Object.

```
model MosJvmExample2
  uses "mosjvm"

  declarations
    o: jvmobject
  end-declarations

  writeln("Creating file")
  o := jvmcallobj("com.fico.examples.FileOperations.createFile")
  if jvmstatus=false then
    setmatherr("Java exception: "+jvmgetexceptionclass)
  end-if
```

```
  writeln("Deleting file")
  jvmcallvoid("com.fico.examples.FileOperations.deleteFile", o)
  if jvmstatus=false then
    setmatherr("Java exception: "+jvmgetexceptionclass)
  end-if

end-model
```

Where the `FileOperations` class is defined as follows:

```
package com.fico.examples;
import java.io.*;

public class FileOperations {
  public static File createFile() throws IOException {
    File f = (new File("testfile.txt")).getAbsoluteFile();
    try (PrintWriter pw = new PrintWriter(new FileWriter(f))) {
      pw.println("Hello World");
    }
    return f;
  }
  public static void deleteFile(File f) throws IOException {
    if (!f.delete()) throw new IOException("Failed to delete file: "+f);
  }
}
```

5.5 Creating and calling methods on Java objects

This example demonstrates calling a Java method that creates an Object, then calls an instance method on that Object.

```
model MosJvmExample2
  uses "mosjvm"

  declarations
    o: jvmobject
  end-declarations

  writeln("Creating File object")
  o := jvmnewobj("java.lang.File","myfile.txt")
  if jvmstatus=false then
    setmatherr("Java exception: "+jvmgetexceptionclass)
  end-if

  writeln("Absolute file path = ", jvmcallstr(o, "getAbsolutePath"))
  if jvmstatus=false then
    setmatherr("Java exception: "+jvmgetexceptionclass)
  end-if

end-model
```

CHAPTER 6

Types

6.1 The jvmobject type

You can use the `jvmobject` type to hold a Java object reference within your Mosel model. A `jvmobject` initially contains a null reference but can be populated as the return value of `jvmcallobj`, e.g.:

```
declarations
  myobj: jvmobject
end-decarations
myobj := jvmcallobj("com.fico.examples.ConfigOperations.getConfigFile","config.dat")
```

Once the `jvmobject` is populated, you can check it's populated using `getisnull` and check if it's of the expected class using `getclass`, e.g.:

```
declarations
  myobj: jvmobject
end-decarations
myobj := jvmcallobj("com.fico.examples.ConfigOperations.getConfigFile","config.dat")
if myobj.isnull then
  writeln('ERROR: Expected configuration File object, found null')
elif myobj.class<>'java.io.File' then
  writeln('ERROR: Expected configuration File object, found '+myobj.class)
end-if
```

You can also turn it to a string or text (which uses the Java `toString()` method), or pass it to other public Java functions, e.g.:

```
declarations
  myobj: jvmobject
  configfiles: array(range) of text
end-decarations
myobj := jvmcallobj("com.fico.examples.ConfigOperations.getConfigFile","config.dat")
configfiles(configfiles.size+1) := text(myobj)
writeln('About to read configuration from file ',myobj)
jvmcallvoid("com.fico.examples.ConfigOperations.loadConfigFromFile",myobj);
```

If the `jvmobject` contains a `java.lang.Number`, a `java.lang.String` or a `java.lang.Boolean`, you can read the value into a Mosel variable, e.g.:

```
declarations
  myobj: jvmobject
  b: boolean
  i: integer
  r: real
  t: text
end-decarations
myobj := jvmcallobj("com.fico.examples.ConfigOperations.getConfigValue","value1")
if myobj.class='java.lang.Integer' then
```

```
  i := myobj.intvalue
elif myobj.class='java.lang.Boolean' then
  b := myobj.boolvalue
elif myobj.class='java.lang.Double' then
  r := myobj.realvalue
elif myobj.class='java.lang.String' then
  t := myobj.textvalue
end-if
```

You can construct a new Java object and store it in a `jvmobject` by using the `jvmnewobj` function, e.g.:

```
declarations
  myobj: jvmobject
end-decarations
myobj := jvmnewobj("java.io.File","C:/myfiles/mydatafile.csv")
```

Finally, you can also populate a `jvmobject` with any of these basic types (the Java object will be created automatically based on the type of the value you provide), e.g.:

```
declarations
  i,b,r,t: jvmobject
end-decarations
i.value := 1
b.value := true
r.value := 9.1
t.value := 'hello world'
```

6.1.1 Attributes

The `jvmobject` type supports attributes allowing different types of object reference to be read/written:

Attribute	Java type	Mosel type
boolvalue	java.lang.Boolean	boolean
bytevalue	java.lang.Byte	integer
charvalue	java.lang.Character	text
floatvalue	java.lang.Float	real
intvalue	java.lang.Integer	integer
longvalue	java.lang.Long	real
realvalue	java.lang.Double	real
shortvalue	java.lang.Short	integer
strvalue	java.lang.String	string
textvalue	java.lang.String	text

CHAPTER 7

Procedures and functions

`getboolvalue`	Retrieve value of a java.lang.Boolean	p. 15
`getbytevalue`	Retrieve value of a java.lang.Byte	p. 16
`getcharvalue`	Retrieve value of a java.lang.Character	p. 17
`getclass`	Retrieve class name from jvmobject	p. 18
`getfloatvalue`	Retrieve value of a java.lang.Float	p. 19
`getintvalue`	Retrieve value of a java.lang.Integer	p. 20
`getisnull`	Query whether a jvmobject contains an object reference	p. 21
`getlongvalue`	Retrieve value of a java.lang.Long	p. 22
`getrealvalue`	Retrieve value of a java.lang.Double	p. 23
`getshortvalue`	Retrieve value of a java.lang.Short	p. 24
`getstrvalue`	Retrieve value of a java.lang.String	p. 25
`gettextvalue`	Retrieve value of a java.lang.String	p. 26
`jvmattachthread`	Attach model thread to JVM	p. 27
`jvmcallint`	Call a Java function that returns an int	p. 28
`jvmcallobj`	Call a Java function that returns an Object	p. 29
`jvmcallreal`	Call a Java function that returns a double	p. 30
`jvmcallstr`	Call a Java function that returns a String	p. 31
`jvmcalltext`	Call a Java function that returns a String	p. 32
`jvmcallvoid`	Call a Java procedure	p. 33
`jvmdetachthread`	Detach model thread from the JVM	p. 34
`jvmgetexceptionclass`	Retrieve type of thrown exception	p. 35
`jvmgetexceptionmsg`	Retrieve message from thrown exception	p. 36
`jvmnewobj`	Call a Java constructor method to create a new Java object	p. 37
`jvmsetcreationargs`	Set arguments to use when creating JVM	p. 38
`jvmstatus`	Query whether last Java method threw exception	p. 39
`setnull`	Clear object reference from jvmobject	p. 40
`setvalue`	Assign object reference stored in jvmobject	p. 41

getboolvalue

Purpose

Retrieve a boolean value held within a `jvmobject`

Synopsis

```
function getboolvalue(obj:jvmobject) : boolean
```

Argument

`obj` The object reference to query, which must be an instance of `java.lang.Boolean`

Return value

The boolean value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a `java.lang.Boolean`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.

Related topics

`setvalue`

getbytevalue

Purpose

Retrieve a byte value held within a `jvmobject`

Synopsis

```
function getbytevalue(obj:jvmobject) : integer
```

Argument

`obj` The object reference to query, which may be any subclass of `java.lang.Number`

Return value

The byte value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a subclass of `java.lang.Number`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.

Related topics

`setvalue`

getcharvalue

Purpose

Retrieve a character value held within a `jvmobject`

Synopsis

```
function getcharvalue(obj:jvmobject) : text
```

Argument

`obj` The object reference to query, which must be an instance of `java.lang.Character`

Return value

The character value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a `java.lang.Character`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.

Related topics

`setvalue`

getclass

Purpose

Query the class of the object referenced by the `jvmobject`

Synopsis

```
function getclass(obj:jvmobject) : string
```

Argument

`obj` The object reference to query

Return value

The name of the object class, e.g. `"java.io.File"`, or an empty string if the `jvmobject` contains a null reference.

Further information

If the reference within the `jvmobject` is null, this function will return an empty string.

getfloatvalue

Purpose

Retrieve a single-precision floating point value held within a `jvmobject`

Synopsis

```
function getfloatvalue(obj:jvmobject) : real
```

Argument

`obj` The object reference to query, which may be any subclass of `java.lang.Number`

Return value

The real value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a subclass of `java.lang.Number`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.

Related topics

`setvalue`

getintvalue

Purpose

Retrieve an integer value held within a `jvmobject`

Synopsis

```
function getintvalue(obj:jvmobject) : integer
```

Argument

`obj` The object reference to query, which may be any subclass of `java.lang.Number`

Return value

The integer value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a subclass of `java.lang.Number`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.

Related topics

`setvalue`

getisnull

Purpose

Query whether the `jvmobject` contains a null object reference

Synopsis

```
function getisnull(obj:jvmobject) : boolean
```

Argument

`obj` The object reference to query

Return value

`true` if the `jvmobject` does not contain a Java object reference, `false` if it references a Java object.

Related topics

`setnull`

getlongvalue

Purpose

Retrieve a long value held within a `jvmobject`

Synopsis

```
function getlongvalue(obj:jvmobject) : real
```

Argument

`obj` The object reference to query, which may be any subclass of `java.lang.Number`

Return value

The long value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a subclass of `java.lang.Number`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.

Related topics

`setvalue`

getrealvalue

Purpose

Retrieve a floating point value held within a `jvmobject`

Synopsis

```
function getrealvalue(obj:jvmobject) : real
```

Argument

`obj` The object reference to query, which may be any subclass of `java.lang.Number`

Return value

The real value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a subclass of `java.lang.Number`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.

Related topics

`setvalue`

getshortvalue

Purpose

Retrieve a short value held within a `jvmobject`

Synopsis

```
function getshortvalue(obj:jvmobject) : integer
```

Argument

`obj` The object reference to query, which may be any subclass of `java.lang.Number`

Return value

The short value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a subclass of `java.lang.Number`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.

Related topics

`setvalue`

getstrvalue

Purpose

Retrieve a string value held within a `jvmobject`

Synopsis

```
function getstrvalue(obj:jvmobject) : string
```

Argument

`obj` The object reference to query, which must be an instance of `java.lang.String`

Return value

The boolean value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a `java.lang.String`, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.
3. If the wish to get the string representation of a `jvmobject` regardless of type, convert to a string directly, ie: `string(obj)` - this will call the Java `toString()` method and will return sensible dummy values if the reference is null or if `toString()` throws an exception.

Related topics

`setvalue`, `gettextvalue`

gettextvalue

Purpose

Retrieve a text value held within a `jvmobject`

Synopsis

```
function gettextvalue(obj:jvmobject) : text
```

Argument

`obj` The object reference to query, which must be an instance of `java.lang.String`

Return value

The boolean value held within the Java object

Further information

1. If the reference within the `jvmobject` is null, calling this function will cause the model to immediately terminate with an error. You can use `getisnull` to check the object reference is set before calling this.
2. If the `jvmobject` is not a java.lang.String, calling this function will cause the model to immediately terminate with an error. You can use `getclass` to check the object reference type before calling this.
3. If the wish to get the text representation of a `jvmobject` regardless of type, convert to a text directly, ie: `text(obj)` - this will call the Java `toString()` method and will return a sensible dummy value if the reference is null.

Related topics

`setvalue`

jvmattachthread

Purpose

Persistently attach the current thread to the Java virtual machine

Synopsis

```
procedure jvmattachthread
```

Further information

1. Normally, a Mosel thread will create a corresponding Java thread the first time it calls a Java method. By calling `jvmattachthread`, you can force the Java thread to be created earlier.
2. 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.
3. Prior to Xpress 8.7, calling `jvmattachthread` was recommended to improve performance. After Xpress 8.7, this is no longer necessary.

Related topics

`jvmdetachthread`

jvmcallint

Purpose

Call a public int function in a Java class.

Synopsis

```
function jvmcallint(qualifiedmethodname:string) : integer
function jvmcallint(qualifiedmethodname:string, ...) : integer
function jvmcallint(targetobject:jvmobject,methodname:string) : integer
function jvmcallint(targetobject:jvmobject,methodname:string, ...) :
      integer
```

Arguments

`qualifiedmethodname`	The name of a static method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`targetobject`	The Java object on which to invoke an instance method
`methodname`	The name of an instance method, not including class and package name, *e.g.* `"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

`jvmstatus`, `jvmdebug`

jvmcallobj

Purpose

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

Synopsis

```
function jvmcallobj(qualifiedmethodname:string) : jvmobject
function jvmcallobj(qualifiedmethodname:string, ...) : jvmobject
function jvmcallobj(targetobject:jvmobject,methodname:string) : jvmobject
function jvmcallobj(targetobject:jvmobject,methodname:string, ...) :
      jvmobject
```

Arguments

`qualifiedmethodname`	The name of the static method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`targetobject`	The Java object on which to invoke an instance method
`methodname`	The name of an instance method, not including class and package name, *e.g.* `"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 any subclass of `java.lang.Object`. 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. You can use `getclass` to check the type of the returned object.
4. The returned `jvmobject` value holds a reference to the Java object returned by the function. This reference will automatically be deleted when the model ends, or the `jvmobject` variable is reset or falls out of scope, or `setvalue` or `setnull` is called for this object.
5. 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

`jvmstatus`, `jvmdebug`, `getclass`, `getisnull`

jvmcallreal

Purpose

Call a public double function in a Java class.

Synopsis

```
function jvmcallreal(qualifiedmethodname:string) : real
function jvmcallreal(qualifiedmethodname:string, ...) : real
function jvmcallreal(targetobject:jvmobject,methodname:string) : real
function jvmcallreal(targetobject:jvmobject,methodname:string, ...) : real
```

Arguments

`qualifiedmethodname`	The name of the static method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`targetobject`	The Java object on which to invoke an instance method
`methodname`	The name of an instance method, not including class and package name, *e.g.* `"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

`jvmstatus`, `jvmdebug`

jvmcallstr

Purpose

Call a public String function in a Java class.

Synopsis

```
function jvmcallstr(qualifiedmethodname:string) : string
function jvmcallstr(qualifiedmethodname:string, ...) : string
function jvmcallstr(targetobject:jvmobject,methodname:string) : string
function jvmcallstr(targetobject:jvmobject,methodname:string, ...) : string
```

Arguments

`qualifiedmethodname`	The name of the static method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`targetobject`	The Java object on which to invoke an instance method
`methodname`	The name of an instance method, not including class and package name, *e.g.* `"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

`jvmstatus`, `jvmdebug`, `jvmcalltext`

jvmcalltext

Purpose

Call a public 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
function jvmcalltext(targetobject:jvmobject,methodname:string) : text
function jvmcalltext(targetobject:jvmobject,methodname:string, ...) : text
```

Arguments

`qualifiedmethodname`	The name of the static method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`targetobject`	The Java object on which to invoke an instance method
`methodname`	The name of an instance method, not including class and package name, *e.g.* `"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

`jvmstatus`, `jvmdebug`, `jvmcallstr`

jvmcallvoid

Purpose

Call a public void function in a Java class.

Synopsis

```
procedure jvmcallvoid(qualifiedmethodname:string)
procedure jvmcallvoid(qualifiedmethodname:string, ...)
procedure jvmcallvoid(targetobject:jvmobject,methodname:string)
procedure jvmcallvoid(targetobject:jvmobject,methodname:string, ...)
```

Arguments

`qualifiedmethodname`	The name of the static method, including class and package name, *e.g.* `"com.fico.examples.MathOperations.multiply"`
`targetobject`	The Java object on which to invoke an instance method
`methodname`	The name of an instance method, not including class and package name, *e.g.* `"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

`jvmstatus`, `jvmdebug`

jvmdetachthread

Purpose

Detach the current thread from the Java virtual machine

Synopsis

```
procedure jvmdetachthread
```

Further information

1. Normally, the Java thread will be destroyed when the Mosel thread exits or the Mosel model terminates. Call `jvmdetachthread` to explicitly detach the thread from the JVM before the end of the model.
2. If you called `jvmattachthread` multiple times from the same system thread, you must call `jvmdetachthread` a corresponding number of times.
3. If you make a Java method call, or call `jvmattachthread`, after called `jvmdetachthread`, a new Java thread will be created for this call.

Related topics

`jvmattachthread`

jvmgetexceptionclass

Purpose

If the last call to a Java method threw an exception, this function will return the exception class name (*e.g.* `java.io.FileNotFoundException`). Otherwise, it will return an empty string.

Synopsis

```
function jvmgetexceptionclass:string
```

Related topics

`jvmstatus`, `jvmgetexceptionmsg`,`jvmdebug`

jvmgetexceptionmsg

Purpose

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

Synopsis

```
function jvmgetexceptionmsg:text
```

Related topics

`jvmstatus`, `jvmgetexceptionclass`,`jvmdebug`

jvmnewobj

Purpose

Call a public constructor in a Java class, returning the newly created object into Mosel as a `jvmobject` value.

Synopsis

```
function jvmnewobj(classname:string) : jvmobject
function jvmnewobj(classname:string, ...) : jvmobject
```

Arguments

`classname`	The name of the Java class to create, *e.g.* `"com.fico.examples.MathOperations"`
`...`	Following the method name, specify zero or more values to pass as arguments to the Java constructor method.

Further information

1. *mosjvm* will look in the class for a constructor with the given argument types. If such a constructor 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. The returned `jvmobject` value holds a reference to the Java object created. This reference will automatically be deleted when the model ends, or the `jvmobject` variable is reset or falls out of scope, or `setvalue` or `setnull` is called for this object.
4. 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

`jvmstatus`, `jvmdebug`

jvmsetcreationargs

Purpose

Set arguments to pass to the Java virtual machine at creation time.

Synopsis

```
procedure jvmsetcreationargs(args:list of string)
procedure jvmsetcreationargs(args:list of text)
```

Argument

`args` List of the arguments to pass to the JVM at startup, *e.g.* `["-Dmy.property.name=hello-world","-Xint"]`

Further information

1. In general, any arguments that could be passed to the `java` executable at the command line can be specified here. Consult the documentation for your Java installation for more details.
2. If a Java virtual machine has already been created in this process, this procedure will do nothing.
3. This procedure can only be used before calling any `jvmcall*,` subroutine, as these will start the in-process JVM.
4. If the NoExec restriction is present, this procedure will cause the model to end with an error.

Related topics

`jvmxmx`, `jvmxms`, `jvmclasspath`

jvmstatus

Purpose

Return whether the last call to a Java method succeeded or threw an exception.

Synopsis

```
function jvmstatus:boolean
```

Further information

1. If the last call to a Java method succeeded, this function will return `true`. If an exception was thrown, it will return `false`.
2. If `jvmstatus` returns `false`, you can get information about the exception thrown using `jvmgetexceptionclass` and `jvmgetexceptionmsg`.

Related topics

`jvmgetexceptionclass`, `jvmgetexceptionmsg`,`jvmabortonexception`

setnull

Purpose

Clear any object reference held within the `jvmobject`.

Synopsis

```
procedure setnull(obj:jvmobject)
```

Argument

`obj` The object reference to clear

Further information

Deletes any reference to a Java object being held by the `jvmobject`. If there are no other references to the Java object, it may become eligible for garbage collection in the usual Java way.

Related topics

`getisnull`

setvalue

Purpose

Set the object reference held within the `jvmobject` to wrap a primitive or string value.

Synopsis

```
procedure setvalue(obj:jvmobject,val:integer)
procedure setvalue(obj:jvmobject,val:real)
procedure setvalue(obj:jvmobject,val:boolean)
procedure setvalue(obj:jvmobject,val:string)
procedure setvalue(obj:jvmobject,val:text)
procedure setboolvalue(obj:jvmobject,val:boolean)
procedure setbytevalue(obj:jvmobject,val:integer)
procedure setcharvalue(obj:jvmobject,val:text)
procedure setfloatvalue(obj:jvmobject,val:real)
procedure setintvalue(obj:jvmobject,val:integer)
procedure setlongvalue(obj:jvmobject,val:real)
procedure setrealvalue(obj:jvmobject,val:real)
procedure setshortvalue(obj:jvmobject,val:integer)
procedure setstrvalue(obj:jvmobject,val:string)
procedure settextvalue(obj:jvmobject,val:text)
```

Arguments

`obj` The object reference to set

`val` The value to store

Further information

1. This will create a Java object to hold the passed value and store its reference within the `jvmobject`. This reference will automatically be deleted when the model ends, or the `jvmobject` variable is reset or falls out of scope, or `setvalue` or `setnull` is called for this object.

2. When calling `setvalue`, the `jvmobject` will be populated with a `java.lang.Integer`, `java.lang.Double`, `java.lang.Boolean` or `java.lang.String`, depending on the type. To specify a different type, call one of the procedures with the type in the name, such as `setshortvalue`.

3. When calling `setcharvalue`, the passed value must be a `text` containing a single character. The model will abort with a runtime error if the value is an empty string or contains multiple characters.

4. Deletes any reference to a Java object that was previously held by the `jvmobject`. If there are no other references to the Java object, it may become eligible for garbage collection in the usual Java way.

5. 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

`getisnull`, `getboolvalue`, `getbytevalue`, `getcharvalue`, `getfloatvalue`, `getintvalue`, `getlongvalue`, `getrealvalue`, `getshortvalue`, `getstrvalue`, `gettextvalue`

CHAPTER 8

Parameters

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

`jvmabortonexception`	Abort model immediately on exception	p. 42
`jvmclasspath`	Classpath to use for JVM	p. 43
`jvmcontainermemoryoptions`	Activate some JVM options to lower memory usage	p. 42
`jvmdebug`	Display stack trace of Java exceptions	p. 43
`jvminterrupt`	Allow Java calls to be interrupted when model is stopped	p. 43
`jvmisloaded`	Check if JVM has been loaded	p. 43
`jvmloadverbose`	Activate additional logging when Java is loaded	p. 44
`jvmxms`	Initial JVM heap size	p. 44
`jvmxmx`	Maximum JVM heap size	p. 44

jvmabortonexception

Description When set to 'true', abort model immediately if a jvmcall* function throw a Java exception.

Type Boolean, read/write

Default value `false`

Note This can be useful if you expect none of your Java calls will throw an exception and you don't want to check `jvmstatus` for each call individually.

jvmcontainermemoryoptions

Description This parameter automatically applies some JVM startup options that will result in lower overall memory usage.

Type Boolean, read/write

Default value `true` when running within DMP, `false` otherwise

Note When `true`, the JVM startup options will be adjusted in an effort to minimize the memory usage of the Java virtual machine, which may be useful when it's competing for resources in a shared environment. However, this will result in more frequent garbage collections which may have a negative impact on performance.

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

jvmclasspath

Description Allows you to set the classpath that the Java virtual machine will use to find classes. If set, this classpath will be used instead of one given in the `CLASSPATH` or `MOSJVM_CLASSPATH` environment variables.

Type String, read/write

Default value `""`

Note When a classpath is specified in this parameter, all components of the classpath must be readable under the currently active Mosel security restrictions.

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.

jvminterrupt

Description True if an in-progress Java call can be interrupted when a model is stopped, false otherwise

Type Boolean, read/write

Default value `true`

Note When the running model is stopped while executing a `jvmcall<type>` function, mosjvm will try to interrupt the Java thread if this parameter is 'true'.

Note Allowing interruptions has a small performance impact which may be noticeable where the user is making hundreds of thousands of very small calls into Java. In these cases, this feature can be disabled by setting the parameter to 'false'.

jvmisloaded

Description True if the current Mosel instance has attached to the JVM, false otherwise

Type Boolean, read only

Default value `false`

Note In general, you can expect this to be 'true' if any model in the current Mosel process has used 'mosjvm' to start the Java virtual machine.

Note If the mosjvm module has been unloaded then reloaded since the JVM was started, this parameter will return 'false' as our link to the JVM has been lost. In this case, calling a Java method from Mosel will re-establish the link to the running JVM.

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.

Note The value of this parameter is shared by all models running in the same Mosel instance

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.

Note The value of this parameter is shared by all models running in the same Mosel instance

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.

Note The value of this parameter is shared by all models running in the same Mosel instance

CHAPTER 9

Deprecated Functionality

9.1 Deprecated Parameters

The functionality described in this chapter is deprecated; it will remain available for the forseeable future but it is not recommended that it be used in new models.

jvmcallstatus

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

Type Integer, read only

Note Deprecated; use `jvmstatus` function instead.

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

Note Deprecated; use `jvmgetexceptionclass` function instead.

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

Note Deprecated; use `jvmgetexceptionmsg` function instead.

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.

Index

A
argument types, 2, 4
arguments, 9, 10
arrays, 2, 5, 10

C
call status, 39
class, 18
CLASSPATH, 3, 43

D
double function call, 30

E
exception class, 35
exception details, 36
exceptions, 4

G
`getboolvalue`, 15
`getbytevalue`, 16
`getcharvalue`, 17
`getclass`, 18
`getfloatvalue`, 19
`getintvalue`, 20
`getisnull`, 21
`getlongvalue`, 22
`getrealvalue`, 23
`getshortvalue`, 24
`getstrvalue`, 25
`gettextvalue`, 26

H
heap size
    initial, 44
    maximum, 44

I
int function call, 28

J
Java command-line arguments, 38
Java search path, 3
JAVA_HOME, 3
JVM, 6
JVM creation options, 38
`jvmabortonexception`, 42
`jvmattachthread`, 27
jvmcall<type>, 2
`jvmcallint`, 28
`jvmcallobj`, 29
`jvmcallreal`, 30
`jvmcallstatus`, 45
`jvmcallstr`, 31
`jvmcalltext`, 32
`jvmcallvoid`, 33
`jvmclasspath`, 43
`jvmcontainermemoryoptions`, 42
jvmdebug, 4
`jvmdebug`, 43
`jvmdetachthread`, 34
`jvmexceptionmsg`, 45
`jvmexceptiontype`, 45
jvmgetexceptionclass, 4
`jvmgetexceptionclass`, 35
jvmgetexceptionmsg, 4
`jvmgetexceptionmsg`, 36
`jvminterrupt`, 43
`jvmisloaded`, 43
`jvmloadverbose`, 44
`jvmnewobj`, 37
jvmobject, 2, 4, 10–12, 15–26, 29, 37, 40, 41
`jvmsetcreationargs`, 38
jvmstatus, 4
`jvmstatus`, 39
`jvmxms`, 44
`jvmxmx`, 44

L
Linux, 6
logging, 44

M
`MOSEL_RESTR`, 7
MOSEL_RESTR, 7
MOSJVM_CLASSPATH, 3
MOSJVM_JAVA_HOME, 3

O
Object constructor call, 37
Object function call, 29

P
primitive value types, 5

R
restrictions, 7
return types, 5

S
setboolvalue, 41
setbytevalue, 41
setcharvalue, 41
setfloatvalue, 41
setintvalue, 41
setlongvalue, 41

`setnull`, 40
setrealvalue, 41
setshortvalue, 41
setstrvalue, 41
settextvalue, 41
`setvalue`, 41
stack trace, 43
String function call, 31, 32
supported platforms, 6
System.err, 5
System.exit, 6
System.out, 5

T
threads, 27, 34

U
unload, 6

V
void function call, 33

W
Windows, 6
workdir, 6
working directory, 6

X
XPRM, 6