BACHELOR THESIS

ooRexx.NET
Bridging .NET and ooRexx

Author
Manuel Raffel

Advisor
ao. Univ. Prof. Mag. Dr. Rony G. Flatscher
Vienna University of Economics and Business

Abstract

This thesis is about the development of ooRexx.NET, a library that enables the usage of .NET classes within the ooRexx programming language.

A brief introduction into several necessary background technologies is given and the details of the implementation of ooRexx.NET are explained. Following this, practical examples show the current possibilities of the library and provide a hands-on approach on how to use the developed library.

The document is concluded with a summary and an outlook for future development directions.

Contents

Part I.
Introduction

This thesis covers the development of ooRexx.NET, a library that enables ooRexx to use the class libraries delivered with Microsoft's .NET Framework.

Open Object Rexx (ooRexx, see Section 1.2) is an object-oriented programming language with roots reaching back as far as 1979. Developed by IBM, it was handed to the open source community in 2004. It includes a C++ API, making it highly extensible, and can be used on most operating systems. It is currently available in version 4.2.0. [1, 2, 3]

The Microsoft .NET Framework (see Section 3) is a software framework prevalently running on computers with the Microsoft Windows operating system. It allows the usage of several programming languages (e.g. C#.NET) to write applications that run on any system which supports the required version of the framework. It was first released in 2002 and has received several updates since then. In July 2015 version 4.6 was released. [16]

OoRexx.NET aims at bridging .NET and ooRexx, making the classes delivered with .NET directly available in ooRexx. To achieve this, it makes heavy use of the BSF4ooRexx (see Section 2) project which is based on the aforementioned C++ API of ooRexx. BSF4ooRexx bridges Java and ooRexx, enabling the latter to use the wealth of Java class libraries as if they were ooRexx objects. [3]

To allow ooRexx to use the classes provided by .NET, ooRexx.NET makes use of the open source project jni4net (see Section 5.3), which provides tools and libraries to use .NET from Java. By combining BSF4ooRexx with jni4net in the library developed for ooRexx.NET (CLR.CLS), it becomes possible to directly work with .NET classes as if they were ooRexx classes.

The following parts of this thesis aim to give an introduction to the technologies used to facilitate the bridge between ooRexx and .NET as well as to demonstrate its abilities. To achieve this, the following content is discussed:

- The main technologies behind the bridge (see Part II).
- Relevant implementation details of ooRexx.NET (see Part III).
- Nutshell examples which demonstrate the usage of ooRexx.NET (see Part IV).

Part II.
Background

To facilitate a deeper understanding for the implementation of ooRexx.NET itself, the following sections introduce the different components which are used directly or indirectly, or have been considered for usage.

1. Open Object Rexx (ooRexx)

Open Object Rexx is an object-oriented programming language that is based on the original REXX (see Section 1.1), which was later extended with object-oriented features (see Section 1.2).

1.1. Restructured Extended Executor (REXX)

The Restructured Extended Executor is an interpreted programming language which was developed in 1979 by Mike F. Cowlishaw for IBM. It was originally intended as a replacement for Exec 2, the batch language used on IBM mainframes at this point in time, but was later defined as the one strategic scripting language on all of IBM's operating systems. [2, 3]

According to Cowlishaw [2], the following fundamental language concepts where applied when the language was designed, some of which are illustrated by the examples shown in Listing 1.1 and Listing 1.2:

- Readability.
- Natural data typing.
- Emphasis on symbolic manipulation.
- Dynamic scoping.
- Nothing to declare.
- System independence.
- Limited-span syntactic units.

- Dealing with reality.
- Adaptability.
- Keep the language small.
- No defined size or shape limits.

```
1 say "Hello World from REXX"
```

Listing 1.1: REXX: A ”Hello World” example

The Rexx Language Association maintains a list (`http://www.rexxla.org/rexxlang/mfc/rexxplat.html`) of the numerous implementations of the language available for current operating systems, examples being Regina (available for Windows, Mac OS X, Unix/Linux, etc.) and REXX/imc (available for several Linux distributions).

1.2. Object Orientation

First thoughts about object-oriented features for REXX have been conducted within IBM at the end of the 1980s, which later led to the development of Object Rexx. It was first distributed in 1997, amongst others for Microsoft Windows. Its source code was handed over to the Rexx Language Association in 2004, which published an open source version named Open Object Rexx (ooRexx) shortly after. [3]

The main benefits of ooRexx include: [1]

- Easy to use and easy to learn.
- Upwardly compatible with classic Rexx.
- The ability to issue commands to multiple environments.
- Offers powerful functions.
- Based on English-like commands.
- Enhanced with full object orientation.
- Designed for object-oriented programming, and also allows Rexx conventional programming.
- Provides a standard Rexx API to develop external function libraries written in C++.

Listing 1.2 shows an example for an object oriented ”Hello World” written in ooRexx.

```
helloworld = .HelloWorld~new()
helloworld~sayHello

::CLASS HelloWorld
  ::METHOD sayHello
    say "Hello World from ooRexx"
```

Listing 1.2: ooRexx: A "Hello World" example

2. BSF4ooRexx

BSF4ooRexx is a package designed to extend the functionality of ooRexx by adding the possibility to directly interact with Java objects. It does so by using the highly extensible C++ API included with ooRexx, thus supplying an external REXX function package and an ooRexx package, named BSF.CLS, to load it. Upon inclusion of BSF.CLS, Java classes can be camouflaged as ooRexx classes. [3]

A short insight into its usage is given in Listing 2.1, which outputs a "Hello World" message by using Java classes and methods.

```
helloworld = bsf.import("java.lang.System")
helloworld~out~println("Hello World from Java (via BSF4ooRexx)")

::requires BSF.CLS
```

Listing 2.1: BSF4ooRexx: A "Hello World" example

3. Microsoft .NET Framework

The Microsoft .NET Framework is a software framework that is usually found on computers using the Microsoft Windows operating system. It consists of a class library known as Framework Class Library (FCL) and a software environment known as Common Language Runtime (CLR), in which the programs are executed. By using a Common Language Infrastructure (CLI), it is possible to write programs using the framework with several different languages (e.g. C#.NET, F#.NET and VisualBasic.NET). They are converted into the Common Intermediate Language (CIL), which is then assembled into so-called bytecode. When the application is executed, the just-in-time compiler of the CLR generates native code out of this bytecode, enabling the program to be executed by the systems processor. [16, 8, 7]

In Listing 3.1, a simple demonstration of the C#.NET programming language, which uses the Microsoft .NET Framework, is given.

```
using System;

class Program
{
  static void Main(string[] args)
  {
    Console.Out.WriteLine("Hello World from C#.NET");
  }
}
```

Listing 3.1: C#.NET: A "Hello World" example

4. Java

Java is a collection of software and specifications that provide a system for developing applications. It consists of several essential components, namely the Java Development Kit (JDK) and the Java Runtime Environment (JRE), itself consisting of the Java Class Library (JCL) and the Java Virtual Machine (JVM). Code for Java can be written in different languages, including the Java programming language itself. The written code is converted into Java bytecode by the Java compiler included in the JDK. The JVM translates the bytecode into native processor instructions at runtime, using the included just-in-time compiler. Therefore, it can be run on any system with the according JRE installed. [13, 14, 10, 12]

In Listing 4.1, a simple demonstration of the Java programming language, which uses the Java platform, is given.

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello World from Java");
  }
}
```

Listing 4.1: Java: A "Hello World" example

5. Bridges

A thoroughly web search conducted in May 2015 revealed three existing bridges that allow using .NET from Java. They differ in the way communication between .NET

and Java was realized, but also in offered features and used licenses. An overview of the found solutions is given in Table 5.1. It lists their name and license, if .NET can be called from Java, if Java can be called from .NET and if they can be used in conjunction with BSF4ooRexx.

Bridge	License	.NET from Java	Java from .NET	BSF4ooRexx
IKVM	zlib	yes	yes	no
javOnet	Commercial	yes	no	yes
jni4net	MIT	yes	yes	yes

Table 5.1: Available Java to .NET bridges

The available bridges are briefly presented in the following sections. All solutions have been investigated in practice and the listed advantages and disadvantages reflect the gathered findings.

5.1. IKVM

IKVM is an implementation of Java for Mono[1] and the Microsoft .NET Framework which includes several components. These are a Java Virtual Machine (JVM), which was implemented in .NET, the Java Class Library (JCL), which was ported to .NET, as well as certain tools that enable interoperability between Java and .NET. [4, 9]

Listing 5.1 shows how to use IKVM to output a ”Hello World” message by using .NET classes and methods.

```
1 public class HelloWorld
2 {
3   public static void main(String[] args)
4   {
5     cli.System.Console.WriteLine("Hello World from .NET (via IKVM)");
6   }
7 }
```

Listing 5.1: IKVM: A ”Hello World” example

The following advantages and disadvantages were found while testing the practical usability of IKVM for ooRexx.NET.

Advantages:

[1] Mono is an open source implementation of Microsoft’s .NET Framework which enables .NET programs to run on other platforms besides Microsoft Windows, e.g. Linux and Mac OS X.

- Support for Mono.

Disadvantages:

- No support for BSF4ooRexx.
- Uses self-implemented JVM.

5.2. javOnet

The commercial interoperability solution javOnet provides a link from Java to .NET by including just one library into the Java project. The library uses a reflection-style programming interface and an internally created .NET process to forward and translate the Java calls on the interface to .NET. [5, 15]

Listing 5.2 shows how to use javOnet to output a ”Hello World” message by using .NET classes and methods.

```
1  import com.javonet.*;
2
3  public class HelloWorld
4  {
5    public static void main(String[] args) throws JavonetException
6    {
7      Javonet.activate("E-MAIL", "ACTIVATION KEY", JavonetFramework.v45);  // commercial license needs
           to be activated
8
9      Javonet.getType("System.Console").invoke("WriteLine", "Hello World from .NET (via javOnet)");
10   }
11 }
```

Listing 5.2: javOnet: A ”Hello World” example

The following advantages and disadvantages were found while testing the practical usability of javOnet for ooRexx.NET.

Advantages:

- Comprehensive documentation.
- One file only.

Disadvantages:

- Commercial.

5.3. jni4net

Jni4net is an open source bridge between Java and .NET, which can be used to access Java from .NET as well as .NET from Java, though for ooRexx.NET only the latter is relevant. It is based on a tool that creates Java proxy classes for specified .NET classes. Using the Java Native Interface[2] (JNI), the methods in the proxy class are linked to specifically generated native libraries that forward the call to real .NET objects. [6]

Listing 5.3 shows how to use jni4net to output a "Hello World" message by using .NET classes and methods.

```
1  import net.sf.jni4net.*;
2
3  public class HelloWorld
4  {
5    public static void main(String[] args)
6    {
7      Bridge.init();
8
9      system.Console.WriteLine("Hello World from .NET (via jni4net)");
10   }
11 }
```

Listing 5.3: jni4net: A "Hello World" example

The following advantages and disadvantages were found while testing the practical usability of jni4net for ooRexx.NET.

Advantages:

- Easy setup.
- Support for BSF4ooRexx.

Disadvantages:

- Very rarely maintained.
- Not well documented.

[2] The Java Native Interface (JNI) enables Java code to call (and be called by) native applications and libraries. [11]

Part III. Implementation

Due to the circumstances explained in Section 5, ooRexx.NET was implemented using jni4net, as it turned out to be the most promising approach. While it is not (yet) able to support Mono, its main advantage is the ability to work with the standard JVM and therefore provide full support for BSF4ooRexx.

Section 6 lists the prerequisites that have to be met in order to make ooRexx.NET work. Section 7 explains the structure and usage of jni4net while Section 8 shows how the capabilities of jni4net had to be extended for ooRexx.NET. In Section 9, the contents of CLR.CLS, the main library of ooRexx.NET, are briefly described.

6. Prerequisites

To be able to work with jni4net and ooRexx.NET in general, the following prerequisites have to be met[3]:

- Oracle Java 8 has to be installed[4].
- Microsoft .NET Framework 4.5.2 has to be installed[5].
- ooRexx 4.2.0 has to be installed[6].
- BSF4ooRexx 4.52 (20150811) has to be installed[7].

7. Using jni4net

Jni4net is available for download at `https://sourceforge.net/projects/jni4net/` and comes in a zipped package containing the files and directories shown in Figure 7.1. To use it within ooRexx, more specifically BSF4ooRexx, the contents of the

[3] The indicated versions denote the version that was used upon implementation and do not necessarily mean that ooRexx.NET will not work with earlier versions of the required components.
[4] Oracle Java can be obtained from `http://www.java.com/download/` (visited on 08/26/2015).
[5] Microsoft .NET Framework can be obtained from `http://www.microsoft.com/download/details.aspx?id=17851` (visited on 08/26/2015).
[6] ooRexx can be obtained from `http://www.oorexx.org/download.html` (visited on 08/26/2015).
[7] BSF4ooRexx can be obtained from `http://sourceforge.net/projects/bsf4oorexx/` (visited on 08/26/2015).

package have to be extracted. Subsequently, the absolute path of the library `jni4net.j-0.8.8.0.jar` has to be added to the systems `CLASSPATH` environment variable[8]. Furthermore, it is necessary to add the main DLL to the Global Assembly Cache[9] (GAC)[10].

```
jni4net-0.8.8.0-bin.zip
├── bin ..... Folder which contains the proxygen tool
│   ├── proxygen.exe ..... Tool to generate Java and .NET proxies
│   ├── proxygen.exe.config ..... Configuration file for proxygen.exe
│   └── ...
├── lib
│   ├── jni4net.j-0.8.8.0.jar ..... Java library which contains basic proxies to .NET
│   ├── jni4net.n.w32.v20-0.8.8.0.dll ..... DLL with proxies for .NET Framework 2.0 32bit
│   ├── jni4net.n.w32.v40-0.8.8.0.dll ..... DLL with proxies for .NET Framework 4.0 32bit
│   ├── jni4net.n.w64.v20-0.8.8.0.dll ..... DLL with proxies for .NET Framework 2.0 64bit
│   ├── jni4net.n.w64.v40-0.8.8.0.dll ..... DLL with proxies for .NET Framework 4.0 64bit
│   ├── jni4net.n-0.8.8.0.dll ..... DLL which autoselects the appropriate DLL
│   └── ...
└── samples ..... Folder which contains several samples.
    └── ...
```

Figure 7.1: jni4net: Contents of jni4net-0.8.8.0-bin.zip

Once the archive is extracted and `CLASSPATH` as well as GAC have been adapted accordingly, jni4net already can be used in ooRexx (see Listing 7.1).

```
1 jni4net = BSF.import("net.sf.jni4net.Bridge")
2
3 jni4net~bsf.dispatch("init")  -- initialize jni4net
```

Listing 7.1: ooRexx: Initializing jni4net

It is worthwhile to note that while BSF4ooRexx is usually able to call Java methods by sending a simple ooRexx message, e.g. `jni4net~init`, in this case it is necessary to explicitly tell BSF4ooRexx to call the Java method `init()` by using `bsf.dispatch("init")`. The reason for this is, that the ooRexx object itself defines a method `init`, which would otherwise be called.

After initialization, the members listed in the tables below can be directly used within ooRexx and provide direct access to the appropriate .NET members[11].

[8] Modifying the `CLASSPATH` environment variable accordingly can be done in a command shell using the command `set CLASSPATH=X:\PATH\TO\LIB\jni4net.j-0.8.8.0.jar;%CLASSPATH%`.

[9] The Global Assembly Cache (GAC) is the central registry of all assemblies available on the system.

[10] Adding the DLL to the GAC can be done by issuing the command `"C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1A\bin\NETFX 4.5.1 Tools\x64\gacutil.exe" /i jni4net.n-0.8.8.0.dll` in a command shell after changing its working directory to the extracted `lib` folder. Note that the path to `gacutil.exe` might be different depending on the installed versions of the .NET Framework.

[11] Contrary to the namespaces used in .NET, those defined by jni4net start with a lowercase letter. For example, the Java member `system.Console` represents the .NET member `System.Console`.

Namespace `system`
`ArgumentException`, `Array`, `AsyncCallback`, `Console`, `DateTime`, `Decimal`, `Delegate`, `Enum`, `Environment`, `Exception`, `Guid`, `IAsyncResult`, `ICloneable`, `IComparable`, `IConvertible`, `IDisposable`, `IFormatProvider`, `IFormattable`, `InvalidOperationException`, `IObject`, `MarshalByRefObject`, `MulticastDelegate`, `NotImplementedException`, `NotSupportedException`, `NullReferenceException`, `Object`, `String`, `SystemException`, `Type`, `ValueType`

Table 7.1: .NET classes of Namespace `system` delivered directly with jni4net

Namespace `system.collections`
`ICollection`, `IDictionary`, `IDictionaryEnumerator`, `IEnumerable`, `IEnumerator`, `IList`

Table 7.2: .NET classes of Namespace `system.collections` delivered directly with jni4net

Namespace `system.io`
`FileStream`, `IOException`, `Stream`, `TextReader`, `TextWriter`

Table 7.3: .NET classes of Namespace `system.io` delivered directly with jni4net

Namespace `system.reflection`
`Assembly`, `BindingFlags`, `ConstructorInfo`, `FieldInfo`, `ICustomAttributeProvider`, `MemberInfo`, `MethodBase`, `MethodInfo`, `ParameterInfo`, `PropertyInfo`

Table 7.4: .NET classes of Namespace `system.reflection` delivered directly with jni4net

Namespace `system.runtime.serialization`
`ISerializable`, `SerializationInfo`, `StreamingContext`

Table 7.5: .NET classes of Namespace `system.runtime.serialization` delivered directly with jni4net

Namespace `system.security`
`IEvidenceFactory`

Table 7.6: .NET classes of Namespace `system.security` delivered directly with jni4net

8. Extending jni4net

As explained in Section 7, jni4net only comes with a limited, i.e. basic, amount of proxy classes to .NET. One possibility to use advanced functionality defined in different assemblies, e.g. Windows Forms, is using reflection at runtime. Another possibility comes with `proxygen.exe`, which is delivered with jni4net in its `bin` folder. This tool has been used in the development of ooRexx.NET to generate proxies to the .NET classes `System.EventHandler` and `System.EventArgs` in order to make them accessible more easily.

The steps necessary to generate the proxy classes are explained in the following sections.

8.1. Configuring proxygen

First, a configuration file has to be created specifying the .NET classes for which proxies should be generated. By putting it directly into the `bin` folder created when jni4net was extracted, its usage is simplified. The content of the file needed to create the two intended proxies can be seen in Listing 8.1. As it is used to create the `oorexx.net.dll` needed for ooRexx.NET, it is named `oorexx.net.proxygen.xml`.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2
3  <jni4net-proxygen xmlns="http://jni4net.sf.net/0.8.8.0/toolConfig.xsd">
4    <TargetDirJvm>java</TargetDirJvm>
5    <TargetDirClr>csharp</TargetDirClr>
6
7    <AssemblyReference
         Assembly="System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
8
9    <ClrType TypeName="System.EventHandler"/>
10   <ClrType TypeName="System.EventArgs"/>
11 </jni4net-proxygen>
```

Listing 8.1: jni4net: Configuration file for `proxygen.exe`

[Lines 4-5] The output folders of the generated files are specified by using the tags `<TargetDirJvm>` and `<TargetDirClr>`.

[Line 7] The assembly in which the needed .NET classes are defined is specified by using the tag `<AssemblyReference>` (can be used multiple times if additional assemblies are needed).

[Lines 9-10] For each .NET class for which a proxy is to be generated, a line with the tag `<ClrType>` is added.

8.2. Invoking proxygen

To invoke `proxygen.exe` with the created configuration file, a command shell needs to be opened and its working directory changed to the `bin` folder of jni4net. The command `proxygen.exe oorexx.net.proxygen.xml` will then generate a `build.cmd` as well as the folders `csharp` and `java`, which contain the generated source code of the proxies.

8.3. Building the Proxies

The generated `build.cmd` contains all instructions necessary to compile the proxies and generate the required libraries to use them. It can be executed using the same command shell used to invoke `proxygen.exe` with the command `build.cmd`.

This invokes both the Java and the .NET compiler to compile the source code generated before. The output is saved in the newly created folder `target`.

8.4. Enabling the Proxies

Of particular importance in the `target` folder are the two libraries `oorexx.net.dll` and `oorexx.net.jar`, which have just been generated. They are best moved into the `lib` folder of the extracted jni4net package. Figure 8.1 shows how the jni4net directory is supposed to look like after all steps above have been carried out.

To enable the freshly generated proxies, the absolute path of the library `oorexx.net.jar` has to be added to the systems `CLASSPATH` environment variable[12]. Subsequently, it is necessary to add the main library to the systems Global Assembly Cache (GAC)[13].

9. CLR.CLS

CLR.CLS is the main library of ooRexx.NET, containing all necessary classes, routines and methods for the bridge to work. The following sections briefly describe the classes

[12] Modifying the `CLASSPATH` environment variable accordingly can be done in a command shell using the command `set CLASSPATH=X:\PATH\TO\LIB\oorexx.net.jar;%CLASSPATH%`.

[13] Adding the DLL to the GAC can be done by issuing the command `"C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1A\bin\NETFX 4.5.1 Tools\x64\gacutil.exe" /i oorexx.net.dll` in a command shell after changing its working directory to jni4net's `lib` folder. Note that the path to `gacutil.exe` might be different depending on the installed versions of the .NET Framework.

```
jni4net-0.8.8.0-bin
├── bin ..... Folder which contains the proxygen tool
│   ├── build.cmd ..... Generated by proxygen.exe
│   ├── oorexx.net.proxygen.xml ..... Configuration file for generating extended proxies
│   ├── proxygen.exe ..... Tool to generate Java and .NET proxies
│   ├── proxygen.exe.config ..... Configuration file for proxygen.exe
│   ├── csharp ..... Folder created upon execution of proxygen.exe
│   ├── java ..... Folder created upon execution of proxygen.exe
│   └── target ..... Folder created upon execution of build.cmd
├── lib
│   ├── jni4net.j-0.8.8.0.jar ..... Java library which contains basic proxies to .NET
│   ├── jni4net.n-0.8.8.0.dll ..... DLL which autoselects the appropriate DLL
│   ├── oorexx.net.jar ..... Java library which contains extended proxies to .NET
│   ├── oorexx.net.dll ..... DLL with extended proxies for appropriate .NET Framework
│   └── ...
└── samples
    └── ...
```

Figure 8.1: jni4net: Files and directories after extending jni4net

and methods contained within this library. A more practical demonstration of the libraries capabilities is given in Part IV.

Upon including CLR.CLS by using the ooRexx `::REQUIRES` directive, the code shown in Listing 9.1 is executed to initialize the bridge.

```
1  #!/usr/bin/rexx
2  /* File:        CLR.CLS
3   * Description: Library for making Microsoft's .NET Framework available in ooRexx: .NET classes and
4   *              objects appear as ooRexx classes and objects and one can send ooRexx messages to them.
5   *              Requires BSF4ooRexx and jni4net.
6   */
7
8  .local~clr.bridge = BSF.import("net.sf.jni4net.Bridge")  -- get jni4net support
9  .clr.bridge~setVerbose(.false)  -- .true for debug output
10
11 .clr.bridge~bsf.dispatch("init")  -- initialize jni4net support
12
13 .local~clr.assembly = BSF.import("system.reflection.Assembly")
14
15 ooRexxAssembly = .clr.assembly~LoadWithPartialName("oorexx.net")  -- get additional .NET proxies
16 .clr.bridge~RegisterAssembly(ooRexxAssembly)
17
18 CALL clr.initAssemblies
19
20 .local~clr.system.array = BSF.import("system.Array")  -- preload frequently used classes
21 .local~clr.system.enum = BSF.import("system.Enum")
22 .local~clr.system.object = BSF.import("system.Object")
23 .local~clr.system.type = BSF.import("system.Type")
24
25 .CLRLogger~setLevel("OFF")  -- turn off logging by default ("OFF")
```

```
26
27 ::REQUIRES BSF.CLS  -- get BSF4ooRexx support
```

Listing 9.1: CLR.CLS: Initialization of the bridge

Apart from initializing the bridge, it is also saving a reference to its static Java class `net.sf.jni4net.Bridge` into the `.local` environment as `.clr.bridge` for further usage. A reference to the static .NET class `system.reflection.Assembly` is also saved as `.clr.assembly` to be used later when necessary. The routine `clr.initAssemblies` is called, its purpose is explained in Section 9.1.1. Furthermore, several frequently used static .NET classes are preloaded. Finally, the default log level is set to `OFF`, to prevent any log output issued by ooRexx.NET in productive environments and BSF4ooRexx support is loaded by including BSF.CLS using the `::REQUIRES` directive of ooRexx.

9.1. Members

CLR.CLS defines several routines which are listed in Table 9.1 in alphabetical order. They are briefly explained in order of their occurrence in CLR.CLS in the following sections.

Name	Modifiers	Parameters	Section
`clr.addAssembly`	PUBLIC	assemblyName	9.1.2
`clr.createArray`	PUBLIC	className, capacity	9.1.7
`clr.createEventHandler`	PUBLIC	rexxEventHandler, rexxData	9.1.6
`clr.extractAssemblyName`	PRIVATE	className	9.1.3
`clr.findAssemblyName`	PRIVATE	className	9.1.4
`clr.findMatchingEvent`	PRIVATE	clrType, eventName	9.1.11
`clr.findMatchingMethod`	PRIVATE	clrType, methodName	9.1.9
`clr.findMatchingProperty`	PRIVATE	clrType, propertyName	9.1.10
`clr.import`	PUBLIC	className	9.1.5
`clr.initAssemblies`	PRIVATE		9.1.1
`clr.wrap`	PRIVATE	param	9.1.8

Table 9.1: Routines defined in CLR.CLS

9.1.1. ::ROUTINE clr.initAssemblies PRIVATE

```
29 /* Routine:     clr.initAssemblies
30  * Description: Stores class names defined in commonly used .NET assemblies ("mscorlib" and "System")
31  *              in the local directory .clr.assemblyName for fast and easy lookup.
```

```
32  */
33 ::ROUTINE clr.initAssemblies PRIVATE
34
35   .local~clr.assemblyName = .directory~new
36
37   CALL clr.addAssembly "mscorlib"
38   CALL clr.addAssembly "System"
```

Listing 9.2: Routine `clr.initAssemblies` in CLR.CLS

While several of the fully qualified class names in .NET directly expose the relevant assembly (e.g. the class `System.Windows.Forms.Form` is defined in the assembly `System.Windows.Forms`), others do not (e.g. the class `System.Security.Cryptography.MD5` is not defined in `System.Security.Cryptography` but in `mscorlib`). This routine creates the local directory `.clr.assemblyName` and calls `clr.addAssembly` (see Section 9.1.2) for the two most basic assemblies, `mscorlib` and `System`, to store the classes contained in them in the created directory for fast and easy lookup.

9.1.2. ::ROUTINE clr.addAssembly PUBLIC

```
40 /* Routine:     clr.addAssembly
41  * Description: Retrieves class names defined in given assembly name and adds them to the local
42  *              directory .clr.assemblyName for fast and easy lookup.
43  * Arguments:   - assemblyName: the assembly name which is to be searched for classes
44  */
45 ::ROUTINE clr.addAssembly PUBLIC
46   PARSE ARG assemblyName
47
48   assemblyTypes = .clr.assembly~LoadWithPartialName(assemblyName)~GetExportedTypes  -- retrieve
         classes defined in assembly with given name
49
50   DO i = 1 TO assemblyTypes~size
51     .clr.assemblyName~setEntry(assemblyTypes[i]~getFullName, assemblyName)
52   END
```

Listing 9.3: Routine `clr.addAssembly` in CLR.CLS

This routine retrieves the fully qualified names of the classes defined in the given `assemblyName`. The names are stored as keys in the local directory `.clr.assemblyName` with the name of the assembly as the value. This is done to speed up the time it takes to find the assembly in which an often needed class is defined in.

9.1.3. ::ROUTINE clr.extractAssemblyName PRIVATE

```
54 /* Routine:     clr.extractAssemblyName
55  * Description: Tries to determine the assembly name of the given class by splitting it and removing
56  *              the last part. Only works for classes where the fully qualified name is just one
57  *              "level" above its assembly name. If the class is in a different assembly, it needs to
```

```
58  *              be added to the known assemblies by calling clr.addAssembly.
59  * Arguments:   - className: the (fully qualified) class name of a .NET class
60  * Returns:     the (supposed) assembly name
61  */
62 ::ROUTINE clr.extractAssemblyName PRIVATE
63   PARSE ARG className
64
65   split = LASTPOS(".", className)
66   assemblyName = SUBSTR(className, 1, (split-1))  -- extract assembly name from fully qualified class
         name
67
68   RETURN assemblyName
```

Listing 9.4: Routine `clr.extractAssemblyName` in CLR.CLS

This routine takes the fully qualified class name `className` as parameter and returns its (supposed) assembly name. This is done by getting the position of the last occurring dot (`.`) and returning the part before it as the assembly name of the given class.

9.1.4. ::ROUTINE clr.findAssemblyName PRIVATE

```
70 /* Routine:     clr.findAssemblyName
71  * Description: Determines the assembly name for the given (fully qualified) class name. If the class
72  *              can not be found in the known assemblies (as instantiated by clr.initAssemblies), the
73  *              (supposed) assembly name is returned as determined by clr.extractAssemblyName.
74  * Arguments:   - className: the (fully qualified) class name of a .NET class
75  * Returns:     the (probably supposed) assembly name
76  */
77 ::ROUTINE clr.findAssemblyName PRIVATE
78   PARSE ARG className
79
80   assemblyName = .clr.assemblyName~entry(className)  -- lookup assembly name in known assemblies
81
82   IF assemblyName <> .nil THEN
83     RETURN assemblyName
84   ELSE
85     RETURN clr.extractAssemblyName(className)
```

Listing 9.5: Routine `clr.findAssemblyName` in CLR.CLS

This routine takes the fully qualified class name `className` as parameter and returns its assembly name. It does so by first checking if the class is defined in the local directory `.clr.assemblyName`, which has been initialized by `clr.initAssemblies` (see Section 9.1.1). If it can not be found there, `clr.extractAssemblyName` (see Section 9.1.3) is called to try to determine the assembly name in another way.

9.1.5. ::ROUTINE clr.import PUBLIC

```
87 /* Routine:     clr.import
88  * Description: Creates and returns an instance of CLRClass which provides a reference to a static
```

```
89  *              .NET class.
90  * Arguments:   - className: the (fully qualified) class name of the static .NET class
91  * Returns:     a CLRClass referencing the given .NET class
92  */
93 ::ROUTINE clr.import PUBLIC
94   PARSE ARG className
95
96   clrClass = .CLRClass~new(className)
97
98   RETURN clrClass
```

Listing 9.6: Routine `clr.import` in CLR.CLS

This routine initializes and returns a new instance of `CLRClass` (see Section 9.3), a .NET proxy class which can handle `static` classes. The parameter `className` is the fully qualified class name of the class.

9.1.6. ::ROUTINE clr.createEventHandler PUBLIC

```
100 /* Routine:     clr.createEventHandler
101  * Description: Makes given ooRexx class inherit from .NET class "System.EventHandler" to enable it to
102  *              receive events triggered by .NET objects.
103  * Arguments:   - rexxEventHandler: the ooRexx class which handles the event
104  *              - rexxData: optional argument to be included in the proxies slotDir~userdata value
105  * Returns:     a Java proxy to the given ooRexx class
106  */
107 ::ROUTINE clr.createEventHandler PUBLIC
108   USE ARG rexxEventHandler, rexxData = .nil
109
110   prxEventHandler = BSFCreateRexxProxy(rexxEventHandler, rexxData)
111   prxClass = bsf.createProxyClass("system.EventHandler")
112   eventHandler = prxClass~new(prxEventHandler)
113
114   RETURN eventHandler
```

Listing 9.7: Routine `clr.createEventHandler` in CLR.CLS

This routine takes the given `rexxEventHandler`, which is an instance of an ooRexx object that provides a method `invoke` handling triggered events, and turns it into a Java proxy class of type `system.EventHandler`. The returned proxy can be assigned to .NET events.

9.1.7. ::ROUTINE clr.createArray PUBLIC

```
116 /* Routine:     clr.createArray
117  * Description: Creates and returns a .NET array of given class and capacity wrapped in a Java proxy.
118  * Arguments:   - className: the type of the objects to be stored in the array
119  *              - capacity: the capacity of the array
120  * Returns:     a .NET array of the given class wrapped in a Java proxy
121  */
122 ::ROUTINE clr.createArray PUBLIC
```

```
123   PARSE ARG className, capacity
124
125   RETURN .clr.system.array~CreateInstance(.clr.system.type~GetType(className), capacity)
```

Listing 9.8: Routine `clr.createArray` in CLR.CLS

This routine creates a Java proxy of type `system.Array` (a .NET array) by supplying the intended type of the array (`className`) and its capacity (`capacity`).

9.1.8. ::ROUTINE clr.wrap PRIVATE

```
127 /* Routine:      clr.wrap
128  * Description: Processes received parameter "param" depending on its type. Always returns a CLR.
129  * Arguments:   - param: any string/object/instance/other which is supposed to be (in) a CLR
130  * Returns:     a CLR instance wrapping "param"
131  */
132 ::ROUTINE clr.wrap PRIVATE
133   USE ARG param
134
135   IF param~isA(.CLR) THEN  -- if "param" already is a CLR, nothing is to be done
136   DO
137     .CLRLogger~trace("clr.wrap: returning unchanged CLR")
138     RETURN param
139   END
140   ELSE IF param~isA(.string) THEN  -- if "param" is a string...
141   DO
142     IF VERIFY(param, "0123456789") = 0 THEN  -- return "System.Int32" if it contains only digits
143     DO
144       .CLRLogger~trace("clr.wrap: new System.Int32 CLR")
145       RETURN .clr~new("System.Int32", param)
146     END
147     ELSE  -- return "System.String" elsewise
148     DO
149       .CLRLogger~trace("clr.wrap: new System.String CLR")
150       RETURN .clr~new("System.String", param)
151     END
152   END
153   ELSE  -- return new CLR, let constructor of CLR handle "param"
154   DO
155     .CLRLogger~trace("clr.wrap: returning new CLR")
156     RETURN .clr~new(param)
157   END
```

Listing 9.9: Routine `clr.wrap` in CLR.CLS

This routine is supposed to always return an object of type `CLR`. Therefore, it takes the parameter `param` and handles it depending on its type. If it is already an object of type `CLR` (see Section 9.2), it is returned unchanged. If the parameter is of type `string`, a new `CLR` of type `System.String` is instantiated and returned. Else, a new `CLR` is created with the unchanged `param` as parameter for further handling.

9.1.9. ::ROUTINE clr.findMatchingMethod PRIVATE

```
159 /* Routine:     clr.findMatchingMethod
160  * Description: Searches given "Type" object for given method name.
161  * Arguments:   - clrType: a .NET "System.Type" defining its available methods
162  *              - methodName: the name of the method to be searched for
163  * Returns:     case sensitive name of the method or .nil if it could not be found
164  */
165 ::ROUTINE clr.findMatchingMethod PRIVATE
166   USE ARG clrType, methodName
167
168   .CLRLogger~trace("Searching method name for" methodName "in" clrType)
169
170   typeMethods = clrType~GetMethods  -- retrieve methods available for this "Type"
171   typeMethodsCount = typeMethods~size
172
173   DO i = 1 TO typeMethodsCount
174     IF typeMethods[i]~getName~caselessEquals(methodName) THEN  -- case insensitive comparison
175       RETURN typeMethods[i]~getName
176   END
177
178   RETURN .nil  -- return .nil only if no matching method was found
```

Listing 9.10: Routine `clr.findMatchingMethod` in CLR.CLS

This routine takes a `System.Type` object `clrType` and a `methodName` as parameters. As the `methodName` is a message caught by any `::method unknown`, it is uppercase. The purpose of this routine is to find the equivalent method name in camel case. This is done by iterating through all the methods supplied by `clrType`, comparing their names to `methodName` and returning the correct one.

9.1.10. ::ROUTINE clr.findMatchingProperty PRIVATE

```
180 /* Routine:     clr.findMatchingProperty
181  * Description: Searches given "Type" object for given property name.
182  * Arguments:   - clrType: a .NET "System.Type" defining its available properties
183  *              - propertyName: the name of the property to be searched for
184  * Returns:     case sensitive name of the property or .nil if it could not be found
185  */
186 ::ROUTINE clr.findMatchingProperty PRIVATE
187   USE ARG clrType, propertyName
188
189   .CLRLogger~trace("Searching property name for" propertyName "in" clrType)
190
191   typeProperties = clrType~GetProperties  -- retrieve properties available for this "Type"
192   typePropertiesCount = typeProperties~size
193
194   DO i = 1 TO typePropertiesCount
195     IF typeProperties[i]~getName~caselessEquals(propertyName) THEN  -- case insensitive comparison
196       RETURN typeProperties[i]~getName
197   END
198
199   RETURN .nil  -- return .nil only if no matching property was found
```

Listing 9.11: Routine `clr.findMatchingProperty` in CLR.CLS

This routine takes a `System.Type` object `clrType` and a `propertyName` as parameters. As the `propertyName` is a message caught by any `::method unknown`, it is uppercase. The purpose of this routine is to find the equivalent property name in camel case. This is done by iterating through all the properties supplied by `clrType`, comparing their names to `propertyName` and returning the correct one.

9.1.11. ::ROUTINE clr.findMatchingEvent PRIVATE

```
201 /* Routine:     clr.findMatchingEvent
202  * Description: Searches given "Type" object for given event name.
203  * Arguments:   - clrType: a .NET "System.Type" defining its available events
204  *              - eventName: the name of the event to be searched for
205  * Returns:     case sensitive name of the event or .nil if it could not be found
206  */
207 ::ROUTINE clr.findMatchingEvent PRIVATE
208   USE ARG clrType, eventName
209
210   .CLRLogger~trace("Searching event name for" eventName "in" clrType)
211
212   typeEvents = clrType~GetEvents  -- retrieve events available for this "Type"
213   typeEventsCount = typeEvents~size
214
215   DO i = 1 TO typeEventsCount
216     IF typeEvents[i]~getName~caselessEquals(eventName) THEN  -- case insensitive comparison
217       RETURN typeEvents[i]~getName
218   END
219
220   RETURN .nil  -- return .nil only if no matching event was found
```

Listing 9.12: Routine `clr.findMatchingEvent` in CLR.CLS

This routine takes a `System.Type` object `clrType` and a `eventName` as parameters. As the `eventName` is a message caught by any `::method unknown`, it is uppercase. The purpose of this routine is to find the equivalent event name in camel case. This is done by iterating through all the methods supplied by `clrType`, comparing their names to `eventName` and returning the correct one.

9.2. ::CLASS CLR PUBLIC

```
222 /* Class:       CLR
223  * Description: Wraps a .NET proxy and handles initialization and method calls to it.
224  */
225 ::CLASS CLR PUBLIC
```

Listing 9.13: Class CLR in CLR.CLS

This class is probably the most important part of ooRexx.NET. It represents an instance of a .NET class and is able to relay ooRexx messages to the according .NET member.

CLR defines several methods which are listed in Table 9.2 in alphabetical order. They are briefly explained in order of their occurrence in CLR.CLS in the following sections.

Name	Modifiers	Parameters	Section
clr.dispatch	PUBLIC	methodName	9.2.3
clr.getInstance	PUBLIC		9.2.6
clr.getType	PUBLIC		9.2.5
clr.setPropertyValue	PRIVATE	propertyName, propertyValue	9.2.7
init	PUBLIC	className, param, ...	9.2.1
string	PUBLIC		9.2.4
unknown	PUBLIC	command, args	9.2.2

Table 9.2: Methods defined by class CLR in CLR.CLS

9.2.1. ::METHOD init PUBLIC

```
227  /* Method:      init
228   * Description: Constructor which creates an instance of CLR based on the supplied class name and
229   *              parameters.
230   * Arguments:   - className: name of the .NET class to be instantiated or a BSF proxy to be wrapped
231   *              - param: first parameter for the class, default .nil
232   *              - ...: additional parameters
233   * Returns:     the freshly created CLR
234   */
235  ::METHOD init PUBLIC
236    EXPOSE clrInstance clrType
237    USE ARG className, param = .nil, ...
238
239    .CLRLogger~trace("Creating new CLR instance of" className "with parameters" param)
240
241    IF className~isA(.string) THEN
242    DO
243      assemblyName = clr.findAssemblyName(className)  -- find assembly of (fully qualified) class name
244
245      SELECT
246        WHEN className = "System.Boolean" THEN  -- special handling for boolean parameters and
             instantiation
247        DO
248          IF param = .true | param~caselessEquals("true") THEN param = "true"
249          ELSE param = "false"
250
251          clrInstance = .clr.assembly~LoadWithPartialName(assemblyName)~CreateInstance(className)
252          clrInstance = clrInstance~GetType~GetMethod("Parse", bsf.createJavaArrayOf(.clr.system.type,
                 .clr.system.type~GetType("System.String")))~Invoke(clrInstance,
                bsf.createJavaArrayOf(.clr.system.object, .clr.bridge~convert(param)))
253        END
254
255        WHEN className = "System.String" THEN  -- special handling for string parameters and
             instantiation
```

```
256         DO
257           clrInstance = .bsf~new("system.String", param)
258         END
259
260         WHEN className = "System.Int32" THEN  -- special handling for integer parameters and
                instantiation
261         DO
262           clrInstance = .clr.assembly~LoadWithPartialName(assemblyName)~CreateInstance(className)
263           clrType = clrInstance~GetType
264           clrInstance = clrType~GetMethod("Parse", bsf.createJavaArrayOf(.clr.system.type,
                  .clr.system.type~GetType("System.String")))~Invoke(clrInstance,
                  bsf.createJavaArrayOf(.clr.system.object, .clr.bridge~convert(param)))
265         END
266
267         OTHERWISE  -- all other classes can be done in an unified way
268         DO
269           argCount = arg() - 1
270
271           IF argCount = 0 THEN  -- no arguments, can use default constructor
272             clrInstance = .clr.assembly~LoadWithPartialName(assemblyName)~CreateInstance(className)
273           ELSE  -- arguments need to be wrapped in an array to be passed to appropriate constructor
274           DO
275             argsList = bsf.createJavaArray(.clr.system.object, argCount)
276
277             DO i = 1 TO argCount
278               .CLRLogger~trace("Parsing argument" arg(i+1))
279               argument = clr.wrap(arg(i+1))
280               .CLRLogger~trace("Parsed argument" argument argument~clr.getType
                     argument~clr.getInstance)
281               argsList[i] = argument~clr.getInstance
282             END
283
284             clrInstance = .clr.assembly~LoadWithPartialName(assemblyName)~CreateInstance(className,
                    .false, .nil, .nil, argsList, .nil, .nil)
285           END
286         END
287       END
288
289       clrType = clrInstance~GetType
290     END
291     ELSE  -- supplied argument is (supposed to be) a BSF_REFERENCE
292     DO
293       clrInstance = className
294       clrType = className~GetType
295     END
296
297     .CLRLogger~trace("Created CLR instance of" className "with clrInstance" clrInstance "and clrType"
          clrType)
```

Listing 9.14: Method `init` in class CLR in CLR.CLS

This method creates a new instance of `CLR` by taking several parameters. The first one, `className`, can either be a `string` denoting the .NET class name for which an instance is to be created, or an instance of a `BSF_REFERENCE` (a BSF4ooRexx Java proxy). In the

latter case, the `CLR` will be constructed out of the existing instance. In the other case, a new proxy will be generated depending on the passed `className` and parameters.

9.2.2. ::METHOD unknown PUBLIC

```
299  /* Method:      unknown
300   * Description: Catches every message sent to this CLR and determines how it is to be treated.
301   * Arguments:   - command: the message sent to this CLR
302   *              - args: the arguments of the message
303   * Returns:     depending on the command
304   */
305  ::METHOD unknown PUBLIC
306    EXPOSE clrInstance clrType
307    USE ARG command, args
308
309    IF RIGHT(command,1) = "=" THEN  -- check if it is an assignment
310    DO
311      IF args[1]~isA(.CLREvent) THEN  -- check passed argument, might already be handled ("+=")
312        RETURN args[1]
313
314      PARSE VAR command property "=" .  -- extract property name (without "=")
315
316      propertyName = clr.findMatchingProperty(clrType, property)  -- try to find given property in
           "Type"
317
318      IF args~items = 1 THEN
319        RETURN self~clr.setPropertyValue(propertyName, args[1])  -- set property value to given
             argument
320    END
321    ELSE  -- not an assignment
322    DO
323      methodName = clr.findMatchingMethod(clrType, command)  -- try to find given method in "Type"
324
325      IF methodName <> .nil THEN  -- aimed for a method
326      DO
327        typeList = bsf.createJavaArray(.clr.system.type, args~items)
328        argsList = bsf.createJavaArray(.clr.system.object, args~items)
329
330        IF args~items > 0 THEN  -- arguments need to be wrapped in an array to be passed to method
331        DO i = 1 TO args~items
332          .CLRLogger~trace("Parsing argument" args[i]~class)
333          argument = clr.wrap(args[i])
334          .CLRLogger~trace("Parsed argument" argument argument~clr.getType argument~clr.getInstance)
335          typeList[i] = argument~clr.getType
336          argsList[i] = argument~clr.getInstance
337        END
338
339        result = clrInstance~GetType~GetMethod(methodName, typeList)~Invoke(clrInstance, argsList)  --
             invoke given method with supplied parameters
340
341        IF result <> .nil THEN
342          RETURN .clr~new(result)
343        ELSE
344          RETURN .nil
```

```
345       END
346       ELSE  -- aimed for another member
347       DO
348         propertyName = clr.findMatchingProperty(clrType, command)  -- try to find given property in
              "Type"
349
350         IF propertyName <> .nil THEN  -- aimed for a property
351         DO
352           propertyInfo = clrInstance~GetType~GetProperty(propertyName)
353
354           RETURN .clr~new(propertyInfo~GetValue(clrInstance, .nil))  -- return value of property
355         END
356         ELSE  -- aimed for an event
357         DO
358           eventName = clr.findMatchingEvent(clrType, command)  -- try to find given event in "Type"
359
360           RETURN .CLREvent~new(self, eventName)  -- return new CLREvent
361         END
362       END
363     END
```

Listing 9.15: Method `unknown` in class CLR in CLR.CLS

This method catches every message sent to this `CLR` (except for methods defined in `CLR` itself, those have to be forcefully forwarded to this method by invoking `clr.dispatch`, see Section 9.2.3). Depending on the supplied `command` and `args`, one of the following steps is performed:

- If the message is an assignment (i.e. code in the form of `command = args[1]` is interpreted) and `args[1]` is an instance of `CLREvent` (see Section 9.4), just return the passed instance of `CLREvent`. This is a special case to handle `+=` calls for assigning event handlers.
- If the message is an assignment, try to find the property name of the wrapped .NET class and set it to the specified value (`args[1]`).
- If the message is not an assignment, try to find a method in the wrapped .NET class which matches the specified `command`. Execute it with the given parameters (`args`) if it could be found and return its result.
- If no appropriate method could be found, try to find a property in the wrapped .NET class which matches the specified `command`. Return its value if it could be found.
- If no appropriate property could be found, assume it is an event and return a new instance of `CLREvent` (see Section 9.4).

9.2.3. ::METHOD clr.dispatch PUBLIC

```
365  /* Method:      clr.dispatch
366   * Description: Directly forwards a message to the "unknown" method of this CLR. This is needed if a
367   *              method on .NET side is to be called which is named like a method of this CLR object.
368   *              Examples are "start" or "init" (inherited from .Object).
369   * Arguments:   - methodName: name of the method to be invoked
370   * Returns:     the return value of the unknown method
371   */
372  ::METHOD clr.dispatch PUBLIC
373    PARSE ARG methodName
374
375    RETURN self~unknown(methodName, arg(2, 'A'))  -- calls unknown method
```

Listing 9.16: Method `clr.dispatch` in class CLR in CLR.CLS

This method needs to be used if one of the methods defined in `CLR` (e.g. `string`) or the ooRexx base class `.Object` (e.g. `start`) is to be invoked on the .NET proxy. Corresponding messages would otherwise be handled by the implemented ooRexx methods rather than by the methods of the .NET class.

9.2.4. ::METHOD string PUBLIC

```
377  /* Method:      string
378   * Description: Overrides "string" method of .Object to enable certain CLR instances to output their
379   *              actual content instead of their class name.
380   * Returns:     the value of the contained proxy
381   */
382  ::METHOD string PUBLIC
383    RETURN self~clr.getInstance~toString
```

Listing 9.17: Method `string` in class CLR in CLR.CLS

This method overrides the `string` method of the ooRexx base class `.Object` to output the actual value (i.e. the output of the `toString()` method defined by every Java object) of the contained .NET object rather than the class name of the `CLR` object.

9.2.5. ::METHOD clr.getType PUBLIC

```
385  /* Method:      clr.getType
386   * Description: Returns the "System.Type" of the .NET object wrapped in this CLR.
387   * Returns:     the "Type" of the .NET object in this CLR
388   */
389  ::METHOD clr.getType PUBLIC
390    EXPOSE clrType
391    RETURN clrType
```

Listing 9.18: Method `clr.getType` in class CLR in CLR.CLS

This method returns the `System.Type` of the contained .NET class.

9.2.6. ::METHOD clr.getInstance PUBLIC

```
393  /* Method:      clr.getInstance
394   * Description: Returns the actual instance of the .NET object wrapped in this CLR.
395   * Returns:     the actual instance of the .NET object in this CLR
396   */
397  ::METHOD clr.getInstance PUBLIC
398    EXPOSE clrInstance
399    RETURN clrInstance
```

Listing 9.19: Method `clr.getInstance` in class CLR in CLR.CLS

This method returns the actual instance of the contained .NET class (a `BSF_REFERENCE` from BSF4ooRexx).

9.2.7. ::METHOD clr.setPropertyValue PRIVATE

```
401  /* Method:      clr.setPropertyValue
402   * Description: Sets the specified property to the supplied value in the .NET object wrapped in
403   *              this CLR.
404   * Arguments:   - propertyName: the name of the property to be set
405   *              - propertyValue: the value the property is to be set to
406   * Returns:     self
407   */
408  ::METHOD clr.setPropertyValue PRIVATE
409    EXPOSE clrInstance clrType
410    USE ARG propertyName, propertyValue
411
412    .CLRLogger~trace("Setting property" propertyName "to" propertyValue)
413
414    IF propertyValue~isA(.CLR) THEN  -- if supplied value is a CLR, extract its wrapped instance
415      propertyValue = propertyValue~clr.getInstance
416    ELSE  -- else handle value according to its (expected) type
417    DO
418      propertyType = clrType~GetProperty(propertyName)~GetPropertyType() -- get expected datatype for
           this property
419
420      IF propertyType~isEnum = .true THEN  -- special handling for enums
421        propertyValue = .clr.system.enum~Parse(propertyType, propertyValue, .true)
422      ELSE
423        propertyValue = .clr~new(propertyType~toString, propertyValue)~clr.getInstance
424    END
425
426    clrType~GetProperty(propertyName)~SetValue(clrInstance, propertyValue, .nil)  -- actually set
       value
427
428    RETURN self
```

Listing 9.20: Method `clr.setPropertyValue` in class CLR in CLR.CLS

This method sets the contained .NET objects property specified by `propertyName` to the value specified by `propertyValue`. It does so by making sure the supplied value is an instance of `CLR`, thereby wrapping a .NET object. If the property expects an `enum` type value, the supplied value is parsed to the according type.

9.3. ::CLASS CLRClass PUBLIC

```
430 /* Class:       CLRClass
431  * Description: Wraps a .NET proxy of a static class and handles its initialization and method calls.
432  */
433 ::CLASS CLRClass PRIVATE
```

Listing 9.21: Class CLRClass in CLR.CLS

This class represents a static .NET class and is able to translate ooRexx messages to the according .NET member.

`CLRClass` defines several methods which are listed in Table 9.3 in alphabetical order. They are briefly explained in order of their occurrence in CLR.CLS in the following sections.

Name	Modifiers	Parameters	Section
`init`	PUBLIC	className	9.3.1
`unknown`	PUBLIC	command, args	9.3.2

Table 9.3: Methods defined by class CLRClass in CLR.CLS

9.3.1. ::METHOD init PUBLIC

```
435   /* Method:      init
436    * Description: Constructor which creates an instance of CLRClass based on the supplied class name.
437    * Arguments:   - className: name of the static .NET class to be referenced
438    * Returns:     the freshly created CLRClass
439    */
440   ::METHOD init PUBLIC
441     EXPOSE clrClass clrType
442     USE ARG className
443
444     .CLRLogger~trace("Creating new CLRClass instance of" className)
445
446     assemblyName = clr.findAssemblyName(className)  -- find assembly of (fully qualified) class name
447
448     clrClass = .clr.assembly~LoadWithPartialName(assemblyName)
449     clrType = clrClass~GetType(className)
450
451     .CLRLogger~trace("Created CLRClass instance of" className "with clrClass" clrClass "and clrType"
        clrType)
```

Listing 9.22: Method `init` in class CLRClass in CLR.CLS

This method creates a new instance of the `CLRClass` by taking the fully qualified class name of a static class as parameter. It then loads the requested assembly and stores a reference to the according type.

9.3.2. ::METHOD unknown PUBLIC

```
453  /* Method:      unknown
454   * Description: Catches every message sent to this CLRClass and determines how it is to be treated.
455   * Arguments:   - command: the message sent to this CLR
456   *              - args: the arguments of the message
457   * Returns:     depending on the command
458   */
459  ::METHOD unknown PUBLIC
460    EXPOSE clrClass clrType
461    USE ARG command, args
462
463    IF RIGHT(command,1) = "=" THEN  -- check if it is an assignment
464    DO
465      IF args[1]~isA(.CLREvent) THEN  -- check passed argument, might already be handled ("+=")
466        RETURN args[1]
467
468      PARSE VAR command property "=" .  -- extract property name (without "=")
469
470      propertyName = clr.findMatchingProperty(clrType, property)  -- try to find given property in
           "Type"
471
472      IF args~items = 1 THEN
473        RETURN self~clr.setPropertyValue(propertyName, args[1])  -- set property value to given
             argument
474    END
475    ELSE
476    DO
477      methodName = clr.findMatchingMethod(clrType, command)  -- try to find given method in "Type"
478
479      IF methodName <> .nil THEN  -- aimed for a method
480      DO
481        typeList = bsf.createJavaArray(.clr.system.type, args~items)
482        argsList = bsf.createJavaArray(.clr.system.object, args~items)
483
484        IF args~items > 0 THEN  -- arguments need to be wrapped in an array to be passed to method
485        DO i = 1 TO args~items
486          .CLRLogger~trace("Parsing argument" args[i]~class)
487          argument = clr.wrap(args[i])
488          .CLRLogger~trace("Parsed argument" argument argument~clr.getType argument~clr.getInstance)
489          typeList[i] = argument~clr.getType
490          argsList[i] = argument~clr.getInstance
491        END
492
493        result = clrType~GetMethod(methodName, typeList)~Invoke(.nil, argsList)  -- invoke given
             method with supplied parameters
494
495        IF result <> .nil THEN
496          RETURN .clr~new(result)
497        ELSE
498          RETURN .nil
499      END
500      ELSE  -- aimed for another member
501      DO
502        propertyName = clr.findMatchingProperty(clrType, command)  -- try to find given property in
             "Type"
```

```
503
504         IF propertyName <> .nil THEN  -- aimed for a property
505         DO
506           propertyInfo = clrType~GetProperty(propertyName)
507
508           RETURN .clr~new(propertyInfo~GetValue(clrType, .nil))  -- return value of property
509         END
510         ELSE  -- aimed for an event
511         DO
512           eventName = clr.findMatchingEvent(clrType, command)  -- try to find given event in "Type"
513
514           RETURN .CLREvent~new(self, eventName)  -- return new CLREvent
515         END
516       END
517     END
```

Listing 9.23: Method unknown in class CLRClass in CLR.CLS

This method is a copy of the `::METHOD unknown` defined in `CLR`, which has been slightly adapted to not need an actual instance of a .NET class.

9.4. ::CLASS CLREvent PRIVATE

```
519 /* Class:       CLREvent
520  * Description: Handles assignment and deassignment of event handlers.
521  */
522 ::CLASS CLREvent PRIVATE
```

Listing 9.24: Class CLREvent in CLR.CLS

This class represents an event defined in a .NET class and is able to add and remove event handlers which are triggered by this event.

`CLREvent` defines several methods which are listed in Table 9.4 in alphabetical order. They are briefly explained in order of their occurrence in CLR.CLS in the following sections.

Name	Modifiers	Parameters	Section
`init`	PUBLIC	clr, eventName	9.4.1
`"+"`	PUBLIC	eventHandler	9.4.2
`"-"`	PUBLIC	eventHandler	9.4.3

Table 9.4: Methods defined by class CLREvent in CLR.CLS

9.4.1. ::METHOD init PUBLIC

```
524  /* Method:      init
525   * Description: Constructor which saves the passed arguments in the class.
526   * Arguments:   - clr: the CLR which wraps the object the event is registered to
527   *              - eventName: the name of the event
528   */
529  ::METHOD init PUBLIC
530    EXPOSE clr eventName
531    USE ARG clr, eventName
```

Listing 9.25: Method `init` in class CLREvent in CLR.CLS

This method creates a new instance of `CLREvent` by saving the passed arguments to local variables.

9.4.2. ::METHOD "+" PUBLIC

```
533  /* Method:      "+"
534   * Description: Is called upon assignment of an event handler with "+=".
535   * Arguments:   - eventHandler: the event handler created by clr.createEventHandler
536   * Returns:     self
537   */
538  ::METHOD "+" PUBLIC
539    EXPOSE clr eventName
540    USE ARG eventHandler
541
542    .CLRLogger~trace("Adding EventHandler " eventHandler "to" clr "(Event:" eventName ")")
543
544    addMethod = "add_" || eventName  -- jni4net reflects the method to add an event handler with
           "add_" and the event name
545    INTERPRET "clr~" || addMethod || "(eventHandler)"  -- let ooRexx carry out the statement in the
           string
546
547    RETURN self
```

Listing 9.26: Method `"+"` in class CLREvent in CLR.CLS

This method handles the assignment of new event handlers (parameter `eventHandler`) to the event specified by this `CLREvent`. It does so by deviating the name of the respective .NET method which adds an event handler from the events name.

9.4.3. ::METHOD "-" PUBLIC

```
549  /* Method:      "-"
550   * Description: Is called upon deassignment of an event handler with "-=".
551   * Arguments:   - eventHandler: the event handler created by clr.createEventHandler
552   * Returns:     self
553   */
554  ::METHOD "-" PUBLIC
555    EXPOSE clr eventName
556    USE ARG eventHandler
```

```
557
558     .CLRLogger~trace("Removing EventHandler " eventHandler "from" clr "(Event:" eventName ")")
559
560     removeMethod = "remove_" || eventName  -- jni4net reflects the method to remove an event handler
            with "remove_" and the event name
561     INTERPRET "clr~" || removeMethod || "(eventHandler)"  -- let ooRexx carry out the statement in the
            string
562
563     RETURN self
```

Listing 9.27: Method `"-"` in class CLREvent in CLR.CLS

This method handles the removal of event handlers (parameter `eventHandler`) from the event specified by this `CLREvent`. It does so by deviating the name of the respective .NET method which removes an event handler from the events name.

9.5. ::CLASS CLRThread PUBLIC

```
565 /* Class:       CLRThread
566  * Description: Provides the environment to start threads interacting with .NET objects. To use it,
567  *              the ooRexx class needs to subclass CLRThread and override the "run" method. It must
568  *              then be started with the "start" method to actually start a new thread.
569  */
570 ::CLASS CLRThread PUBLIC
```

Listing 9.28: Class CLRThread in CLR.CLS

This class acts as the base class for ooRexx classes required to run in a separate thread and wishing to access .NET objects in other threads.

`CLRThread` defines several methods which are listed in Table 9.5 in alphabetical order. They are briefly explained in order of their occurrence in CLR.CLS in the following sections.

Name	Modifiers	Parameters	Section
`run`	ABSTRACT		9.5.2
`start`	PUBLIC	rexxData	9.5.1

Table 9.5: Methods defined by class CLREvent in CLR.CLS

9.5.1. ::METHOD start PUBLIC

```
572   /* Method:      start
573    * Description: Creates a new thread which is able to flawlessly interact with .NET objects in
574    *              different threads. Must be called to execute the "run" method in a new thread.
575    * Arguments:   - rexxData: optional argument to be included in the proxies slotDir~userdata value
576    */
```

```
577   ::METHOD start PUBLIC
578     USE ARG rexxData = .nil
579     rexxRunnableProxy = BsfCreateRexxProxy(self, rexxData, "java.lang.Runnable")
580     rexxThread = .bsf~new("java.lang.Thread", rexxRunnableProxy)
581     rexxThread~bsf.dispatch("start")
```

Listing 9.29: Method `start` in class CLRThread in CLR.CLS

This method starts a new `java.lang.Thread` which is executed in parallel to other threads. It does so by using the classes `run` method to overwrite the `public void run()` method of `java.lang.Runnable`. Therefore, invoking `start` on an ooRexx class inheriting from `CLRThread` will run the classes `run` method in a new Java thread.

9.5.2. ::METHOD run ABSTRACT

```
583   /* Method:      run
584    * Description: Needs to be overridden by specialising ooRexx class.
585    */
586   ::METHOD run ABSTRACT
```

Listing 9.30: Method `run` in class CLRThread in CLR.CLS

This method is supposed to be overwritten by ooRexx classes inheriting from `CLRThread`.

9.6. ::CLASS CLRLogger PUBLIC

```
588 /* Class:       CLRLogger
589  * Description: Provides simple logging functionality within CLR.CLS and for programs using it.
590  *              Different log levels are defined and can be used to output all kinds of information
591  *              at runtime or for debugging purposes. The kind of messages displayed can be limited
592  *              by specifying the intended log level.
593  */
594 ::CLASS CLRLogger PUBLIC
```

Listing 9.31: Class CLRLogger in CLR.CLS

The class `CLRLogger` is a convenience class built to enable multiple log levels within CLR.CLS and programs using this library.

`CLRLogger` defines several methods which are listed in Table 9.6 in alphabetical order. They are briefly explained in order of their occurrence in CLR.CLS in the following sections.

Name	Modifiers	Parameters	Section
`init`	PUBLIC CLASS		9.6.1
`output`	PRIVATE CLASS	logLevel, message	9.6.4
`setLevel`	PUBLIC CLASS	desiredLevel	9.6.2
`unknown`	PUBLIC CLASS	levelName, args	9.6.3

Table 9.6: Methods defined by class CLRLogger in CLR.CLS

9.6.1. ::METHOD init PUBLIC CLASS

```
596  /* Method:      init
597   * Description: Constructor which defines the available log levels.
598   */
599  ::METHOD init PUBLIC CLASS
600    EXPOSE logLevels
601
602    logLevels = .directory~new
603
604    logLevels["OFF"] = 70
605    logLevels["FATAL"] = 60
606    logLevels["ERROR"] = 50
607    logLevels["WARN"] = 40
608    logLevels["INFO"] = 30
609    logLevels["DEBUG"] = 20
610    logLevels["TRACE"] = 10
```

Listing 9.32: Method `init` in class CLRLogger in CLR.CLS

This method initializes the class by defining the following log levels and storing them in the directory `logLevels`:

- OFF
- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

9.6.2. ::METHOD setLevel PUBLIC CLASS

```
612  /* Method:      setLevel
613   * Description: Sets the log level limit. Only messages of this type (and above) will be outputted.
614   *              For example, a log level limit of "ERROR" will only output messages of type
615   *              "ERROR" and "FATAL".
616   * Arguments:   - desiredLevel: the log level limit
617   */
```

```
618  ::METHOD setLevel PUBLIC CLASS
619    EXPOSE logLevel logLevels
620    PARSE ARG desiredLevel
621
622    IF logLevels[desiredLevel] <> .nil THEN  -- check if supplied argument is a valid log level
623      logLevel = logLevels[desiredLevel]
624    ELSE
625      logLevel = logLevels["OFF"]
```

Listing 9.33: Method `setLevel` in class CLRLogger in CLR.CLS

Calling this method sets the threshold for the output of log messages. E.g., a threshold level of `INFO` will output only messages of level `INFO` and above, i.e. `WARN`, `ERROR` and `FATAL`, but not `DEBUG` and `TRACE`.

9.6.3. ::METHOD unknown PUBLIC CLASS

```
627  /* Method:      unknown
628   * Description: Catches log messages. For example, a log output of type "INFO" can be invoked by
629   *              .CLRLogger~info("log output"). Output is given (or not) depending on the set level
630   *              limit.
631   * Arguments:   - levelName: the log level this message belongs to
632   *              - args: the message itself
633   */
634  ::METHOD unknown PUBLIC CLASS
635    EXPOSE logLevel logLevels
636    USE ARG levelName, args
637
638    IF args~ITEMS > 0 & logLevels[levelName] <> .nil THEN  -- check if there is a message and if the
         given log level is defined
639    DO
640      message = args[1]
641
642      IF logLevels[levelName] >= logLevel THEN  -- only process message if invoked log level is
           greater or equal log level limit
643        .CLRLogger~output(levelName, message)
644    END
```

Listing 9.34: Method `unknown` in class CLRLogger in CLR.CLS

This method listens for any message sent to `CLRLogger`. It checks, if the supplied command is one of the log levels defined in the `logLevels` directory. If it is, and if its value is above the defined threshold, the method `output` is called.

9.6.4. ::METHOD output PRIVATE CLASS

```
646  /* Method:      output
647   * Description: Provides formatting and actual output for a log message.
648   * Arguments:   - logLevel: the log level this message belongs to
649   *              - message: the message itself
650   */
```

```
651   ::METHOD output PRIVATE CLASS
652     PARSE ARG logLevel, message
653
654     SAY "[" || .dateTime~new || " | " ||  LEFT(logLevel, 5) || "] :: " || message  -- default output
          with date, time, log level (always five characters for consistent columns) and message
```

Listing 9.35: Method `output` in class CLRLogger in CLR.CLS

This method prints the supplied message attached to the current date, time and the used log level.

Part IV.
Practical Usage

The following sections show five examples which were written using ooRexx.NET. They are intended to provide a practical approach on how to use this library and also show what can be done with it.

For the sake of convenience, the code of the samples is split into cohesive chunks and elucidated step by step. The continuous version of each examples code can be found in Section E of the Appendix. For a reference of the used .NET classes and methods, see `https://msdn.microsoft.com/`.

10. Example 1: Hello World

```
1 #!/usr/bin/rexx
2 /* File:        01-helloworld-clr.rexp
3  * Description: Simple output of "Hello World" via .NET classes.
4  * Shows:       - Import of static classes
5  *              - Invokation of methods (with parameters)
6  */
```

The first example (see Listing E.1) performs the traditional output of ”Hello World” on the console by using a .NET class and method. This is done by getting a reference to the appropriate .NET class and subsequently calling the method which outputs a string to the console in .NET.

```
8 console = clr.import("System.Console")  -- get reference to static class "System.Console"
```

[Line 8] The first statement imports the static .NET class `System.Console` by calling `clr.import` (see Section 9.1.5). The returned object is a instance of CLRClass (see Section 9.3), which internally references the specified .NET class. In .NET, the imported class handles, amongst other things, the in- and output from and to the console.

```
9 console~WriteLine("Hello World from ooRexx.NET (via BSF4ooRexx and jni4net)")  -- invoke method
    "WriteLine" with parameter "Test"
```

[Line 9] Now that the CLRClass is assigned to the variable `console`, it can be used like any other ooRexx object. `System.Console` defines the method `static void WriteLine( string value)`, which can now be called directly from ooRexx by sending a message to the CLR object. Case sensitivity is thereby not important - the same method could also be called with the command `console~wRiTeLinE("[...]")` and would still be executed correctly.

```
11 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

[Line 11] Finally, ooRexx.NET is included using the `::REQUIRES` directive of ooRexx.

Execution of the code above yields the console output shown below:

```
1 Hello World from ooRexx.NET (via BSF4ooRexx and jni4net)
```

11. Example 2: Event Log

```
1 #!/usr/bin/rexx
2 /* File:        02-eventlog-clr.rexp
3  * Description: Reads the "Application" log of the current machine and outputs every entry. Uses the
4  *              "System.Media.SystemSounds" class to emit a beep for every written message.
5  * Shows:       - Import of static classes
6  *              - Instantiation of classes (with parameters)
7  *              - Retrieval of properties
8  *              - Invokation of methods (without parameters)
9  */
```

The second example (see Listing E.2) demonstrates two potentially interesting possibilities of working with .NET from ooRexx. Sound output of system specific sounds is generated by importing the relevant class and issuing a certain command. Additionally, the systems internal log is accessed and outputted by instantiating a .NET class and enumerating over the messages stored within.

```
11 systemSounds = clr.import("System.Media.SystemSounds")  -- get reference to static class
       "System.Media.SystemSounds"
```

[Line 11] The first statement imports the static .NET class `System.Media.SystemSounds` and stores the resulting CLRClass as `systemSounds`. In .NET, the imported class encapsulates different sounds associated with a set of Windows operating system events, e.g. the sound that is played when an error message is shown.

```
13 eventLog = .clr~new("System.Diagnostics.EventLog", "Application")  -- create instance of class
       "System.Diagnostics.EventLog" with parameter "Application"
14 logEvents = eventLog~Entries~GetEnumerator  -- retrieve log entry collection, create enumerator
```

[Line 13] A new CLR object (see Section 9.2) with the parameters specified in parentheses is instantiated. The first parameter is used as the class name of the .NET class to be instantiated. The second parameter is passed to the constructor of the specified class. The returned CLR encapsulates the `System.Diagnostics.EventLog` class which can be used to create new or retrieve existing system event logs. By passing its constructor the parameter "Application", the existing log named "Application" is retrieved and henceforth accessible as ooRexx variable `eventLog`.

[Line 14] The message sent to `eventLog` consists of two parts. The first part retrieves the .NET property "Entries", which is a `EventLogEntryCollection`, wrapped in a CLS. The second part calls the method `public IEnumerator GetEnumerator()` to retrieve an implementation of the .NET interface `IEnumerator` which is able to iterate through the single entries stored inside `Entries`.

```
16 DO WHILE logEvents~MoveNext <> .false  -- iterate over log entries
17   eventEntry = logEvents~Current
18   systemSounds~Beep~Play  -- play system sound "beep"
19   SAY eventEntry~TimeGenerated "::" eventEntry~Message  -- output time and message of event
20 END
```

[Line 16] The command `logEvents~MoveNext` calls the .NET method `bool Move Next()`, which advances the enumerator to the next element of the collection. By combining it with the ooRexx `DO WHILE` loop, the instructions within the block will be repeated for each element in the collection.

[Lines 17–19] After retrieving the `System.Diagnostics.EventLogEntry` from the current position in the enumerator, the system sound "Beep" is played by calling the .NET method `public void Play()`. Finally, the retrieved log message is written to the console by using the ooRexx command `SAY`. Both the time the log entry was generated and the log entry itself are outputted by retrieving the values of the .NET properties `TimeGenerated` and `Message`.

```
22 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

[Line 22] Finally, ooRexx.NET is included using the `::REQUIRES` directive of ooRexx.

Execution of the code above yields similar console output to that shown below:

```
[...]
20.01.2015 17:25:59 :: Expected event (Stream service as user abnormally ended. Re-launching... [0]).
20.01.2015 17:26:00 :: Expected event (SSAU process launched 8944 [0]).
20.01.2015 17:26:00 :: Expected event (Stream service as user abnormally ended. Re-launching... [0]).
20.01.2015 17:26:01 :: Expected event (SSAU process launched 7516 [0]).
20.01.2015 17:26:01 :: Expected event (Stream service as user abnormally ended. Re-launching... [0]).
20.01.2015 17:26:02 :: Expected event (SSAU process launched 9948 [0]).
20.01.2015 17:26:02 :: Expected event (Stream service as user abnormally ended. Re-launching... [0]).
20.01.2015 17:26:03 :: Expected event (SSAU process launched 10012 [0]).
20.01.2015 17:26:03 :: Expected event (Stream service as user abnormally ended. Re-launching... [0]).
20.01.2015 17:26:04 :: Expected event (SSAU process launched 3892 [0]).
[...]
```

12. Example 3: System Events

```
#!/usr/bin/rexx
/* File:        03-systemevents-clr.rexp
 * Description: Reacts to a Windows system event, namely a change of the systems clock.
 * Shows:       - Import of static classes
 *              - Instantiation of classes (without parameters)
 *              - Creation and registration of event handlers
 */
```

The third example (see Listing E.3) demonstrates how ooRexx can be hooked up to system events by the use of ooRexx.NET. This is done by importing the responsible .NET class and attaching an event handler to the event that is to be reacted upon.

```
9  systemEvents = clr.import("Microsoft.Win32.SystemEvents")  -- get reference to static class
       "Microsoft.Win32.SystemEvents"
```

[Line 9] The static .NET class `Microsoft.Win32.SystemEvents` is imported and the resulting CLRClass is stored as `systemEvents`. In .NET, the imported class offers several events triggered by certain changes in the systems environment, e.g. `Display SettingsChanged`, occurring when the screen resolution is changed, or `SessionEnding`, occurring when the system is about to log off or shut down.

```
11 systemEventHandler = clr.createEventHandler(.SystemEventHandler~new)  -- create new event handler from
       ooRexx class "SystemEventHandler"
12 systemEvents~TimeChanged += systemEventHandler  -- register event handler to "TimeChanged" event
```

[Line 11] By using the method `clr.createEventHandler` coming with ooRexx.NET (see Section 9.1.6), the new instance of SystemEventHandler (see below for an explanation of this class) is internally transformed into an .NET event handler and saved as `systemEventHandler`.

[Line 12] The `+=` operator registers the created event handler to the `TimeChanged` event. From now on, events of the specified type are forwarded to its `invoke` method.

```
14 SAY "Waiting for ""TimeChanged"" event..."
15 PARSE PULL  -- wait for user input to prevent program from closing
```

[Lines 14-15] Using the ooRexx command `PARSE PULL` is an easy possibility to ensure the program is not quit after processing all commands and can listen for the registered event.

```
17 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

[Line 17] ooRexx.NET is included using the `::REQUIRES` directive of ooRexx.

```
19 /* Class:       SystemEventHandler
20  * Description: Implements the .NET event handler invoked by the "TimeChanged" event.
21  */
22 ::CLASS SystemEventHandler
23
24   /* Method:      invoke
25    * Description: Called by .NET when the registered event is triggered.
26    * Arguments:   - caller: the object the event handler was registered to
27    *              - eventArgs: the event arguments passed by the caller
28    */
29   ::METHOD invoke
30     USE ARG caller, eventArgs
31
32     SAY "Windows clock has been changed!"
```

[Line 22] For ooRexx to be able to listen for events offered by the specified .NET class, it is necessary to write a class which is able to handle those events.

[Lines 29-32] The ooRexx event handler class needs to implement a method `invoke` which takes two parameters, the object triggering the event and the arguments coming with it. In this example, a simple message is written to the console when the handler is triggered.

Execution of the code above yields the console output shown below:

```
Waiting for "TimeChanged" event...
```

Upon manual changing of the system clock, the event is triggered and the output changes to:

```
Waiting for "TimeChanged" event...
Windows clock has been changed!
```

13. Example 4: Forms

```
#!/usr/bin/rexx
/* File:        04-forms-clr.rexp
 * Description: Creates a windows form with a progress bar and a start button which, when clicked,
 *              starts a new thread to increase the progress bar.
 * Shows:       - Instantiation of classes (without parameters)
 *              - Manipulation of properties
 *              - Invokation of methods (with parameters)
 *              - Creation and registration of event handlers
 *              - Import of static classes
 */
```

The fourth example (see Listing E.4) demonstrates the usage of .NET forms combined with threads and event handlers. Several controls are initialized and combined to show a simple windows form (see Figure 13.1). An event handler is attached to the button labeled ”Start”, which triggers a new thread to increase the value of the shown progress bar.

```
winForm = .clr~new("System.Windows.Forms.Form")  -- create instance of class
    "System.Windows.Forms.Form"
winForm~Text = "Processor"  -- set property "Text" to string "Processor"
winForm~AutoSize = .true  -- set property "AutoSize" to boolean true
winForm~AutoSizeMode = GrowAndShrink  -- set property "AutoSizeMode" to enumeration value
    "GrowAndShrink"
```

[Lines 12–15] The first part of the program instantiates a new `System.Windows.Forms.Form`, the .NET class used for user interfaces. Several properties, i.e. its title (the property `Text`) and its sizing behaviors (the properties `AutoSize` and `AutoSizeMode`), are set. The last property is especially noteworthy, as it expects a value of type `System.Windows.Forms.AutoSizeMode`, which is an enumeration offering the two values `AutoSizeMode.GrowAndShrink` and `AutoSizeMode.GrowOnly`. ooRexx.NET automatically does the conversion to the appropriate .NET type, so that it

is enough to just provide the name part of the intended enumeration value.

```
17 contentPane = .clr~new("System.Windows.Forms.FlowLayoutPanel")
18 contentPane~AutoSize = .true  -- set property "AutoSize" to boolean true
19 contentPane~AutoSizeMode = GrowAndShrink  -- set property "AutoSizeMode" to enumeration value
       "GrowAndShrink"
20 winForm~Controls~Add(contentPane)  -- add "FlowLayoutPanel" to "Form"
```

[Lines 17-19] This part of the program instantiates a new `System.Windows.Forms.FlowLayoutPanel`, a .NET panel which automatically arranges the controls contained within. Its sizing behavior is set accordingly to its parent form, making sure the control uses as little space as necessary.

[Line 20] The created panel is added to the forms `Controls` property, a collection of all its child controls.

```
22 progressBar = .clr~new("System.Windows.Forms.ProgressBar")
23 progressBar~Minimum = 0  -- set property "Minimum" to integer 0
24 progressBar~Maximum = 100  -- set property "Maximum" to integer 100
25 progressBar~Value = 0  -- set property "Value" to integer 0
26 contentPane~Controls~Add(progressBar)  -- add "ProgressBar" to "FlowLayoutPanel"
```

[Lines 22-26] This third part of the program instantiates a new `System.Windows.Forms.ProgressBar`. The `ProgressBar` is a .NET control which can, for example, be used to show to what extent a certain task has already been processed. After setting several properties, it is added to the `FlowLayoutPanel` instantiated before to make it the first element to be shown in the form.

```
28 startButton = .clr~new("System.Windows.Forms.Button")
29 startButton~Text = "Start"  -- set property "Text" to string "Start"
30 contentPane~Controls~Add(startButton)  -- add "Button" to "FlowLayoutPanel"
```

[Lines 28-30] This part of the program instantiates a new `System.Windows.Forms.Button`, the .NET class representing a simple button to click on. Its properties are set before it is added to the forms layout panel.

```
32 mouseEventHandler = clr.createEventHandler(.MouseEventHandler~new(progressBar, startButton))  --
       create new event handler from ooRexx class "MouseEventHandler"
33 startButton~Click += mouseEventHandler  -- register event handler to "Click" event
```

[Lines 32-33] A new instance of the ooRexx class MouseEventHandler (see below for an explanation of this class) is created and gets the progress bar and the start button as parameters. It is wrapped into an .NET event handler by the method `clr.createEventHandler` and registered to the button's `Click` event.

```
35 application = clr.import("System.Windows.Forms.Application")  -- get reference to static class
       "System.Windows.Forms.Application"
36 application~Run(winForm)  -- invoke method "Run", which starts an application message loop
```

[Lines 35-36] The first line imports the static .NET class `System.Windows.Forms.Application`. A a new application message loop which handles the created form is started thereafter by invoking the `public static void Run(Form mainForm)` method defined in the class.

```
38 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

[Line 38] ooRexx.NET is included using the `::REQUIRES` directive of ooRexx.

```
40 /* Class:       MouseEventHandler
41  * Description: Implements the .NET event handler invoked by the "Click" event.
42  */
43 ::CLASS MouseEventHandler
44
45   /* Method:      init
46    * Description: Constructor which saves the passed arguments in the class.
47    * Arguments:   - progressBar: the progress bar which is to be modified
48    *              - startButton: the start button which is to be modified
49    */
50   ::METHOD init
51     EXPOSE progressBar startButton
52     USE ARG progressBar, startButton
53
54   /* Method:      invoke
55    * Description: Called by .NET when the registered event is triggered.
56    * Arguments:   - caller: the object the event handler was registered to
57    *              - mouseEventArgs: the event arguments passed by the caller
58    */
59   ::METHOD invoke
60     EXPOSE progressBar startButton
61     USE ARG caller, mouseEventArgs
62
63     .Processor~new(progressBar, startButton)~start  -- creates a new instance of ooRexx class
         "Processor", which is a thread, and starts it
```

[Line 43] For ooRexx to be able to listen for events offered by the specified .NET class, it is necessary to write a class which is able to handle those events.

[Lines 50-52] Upon instantiation of this class, the two parameters `progressBar` and `startButton` are stored within the class under the same name.

[Lines 59-63] The ooRexx event handler class needs to implement a method `invoke` which takes two parameters, the object triggering the event and the arguments coming with it. In this example, it instantiates a new ooRexx Processor instance (which is explained below) and invokes its `start` method. It further uses the two local variables `progressBar` and `startButton` as parameter for the new instance.

```
65 /* Class:       Processor
66  * Description: Inherits from CLRThread defined in CLR.CLS, which enables this class to run in its own
67  *              thread. The "run" method has to be implemented, as it is the method executed upon
68  *              start of the thread. The thread must be started by invoking "start", which is a method
69  *              implemented in CLRThread. Invoking "run" will not start a concurrent execution of the
70  *              method!
71  */
72 ::CLASS Processor SUBCLASS CLRThread
73
74   /* Method:      init
75    * Description: Constructor which saves the passed arguments in the class.
76    * Arguments:   - progressBar: the progress bar which is to be modified
77    *              - startButton: the start button which is to be modified
78    */
79   ::METHOD init
80     EXPOSE progressBar startButton
81     USE ARG progressBar, startButton
82
83   /* Method:      run
84    * Description: Executed upon start of the thread (by invoking "start").
85    */
86   ::METHOD run
87     EXPOSE progressBar startButton
88
89     startButton~Enabled = .false  -- disable start button to prevent multiple clicks
90
91     DO i = 1 TO 100
92       progressBar~Value = i  -- set value of the progress bar (from 1 to 100)
93       CALL syssleep .1  -- sleep 100 milliseconds to prevent reaching 100 immediately
94     END
95
96     startButton~Text = "Finished"  -- set text on button to "Finished"
```

[Line 72] The class Processor is a subclass of CLRThread, which is included in ooRexx.NET. The base class specifies a method `start` which takes care of executing the `run` method of its specializing class in its own thread. The thread is able to access and manipulate objects in the .NET applications message loop.

[Lines 79-81] Upon instantiation of this class, the two parameters `progressBar` and `startButton` are stored within the class under the same name.

[Lines 86-87] The `run` method is the one actually carrying out the work in the thread started by invoking the `start` method on the base class CLRThread.

[Line 89] The start button is disabled to prevent the starting of multiple threads.

[Lines 91-94] The progress bar's value is increased step by step from 1 to 100. The thread is put to sleep for one tenth of a second after each increase, to slow down the process and make it visible for the human eye.

[Line 96] After the progress bar's value has reached 100, the start button's text is changed to "Finished" to indicate that the thread carried out its work.

Execution of the code above shows the form depicted in Figure 13.1

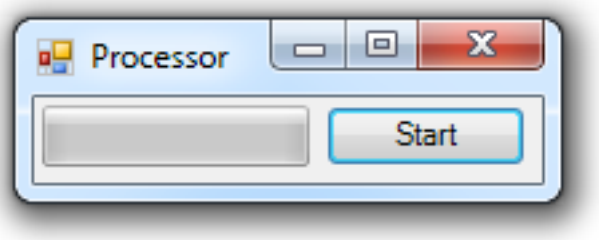

Figure 13.1: Example 4: A windows form created by ooRexx.

Upon clicking on the button labeled "Start", it is disabled and the created thread starts increasing the progress bar (see Figure 13.2).

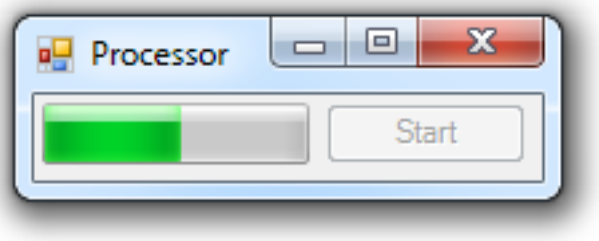

Figure 13.2: Example 4: The form after pushing the "Start" button.

Once the progress bar has reached its maximal value, the text in the button changes to "Finished" (see Figure 13.3).

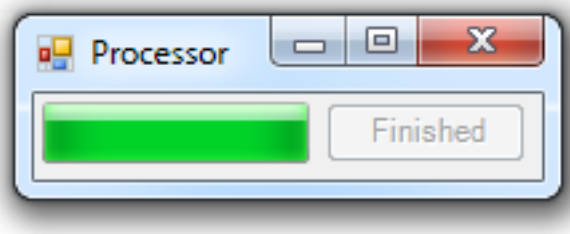

Figure 13.3: Example 4: The form after its "Processor" finished working.

14. Example 5: Server/Client

The fifth and last example shows a simple server/client communication realized with .NET classes. The following two sections explain the two main components, the server (Section 14.1) and the client (Section 14.2), which are realized in different ooRexx programs and use a TCP connection to communicate with each other. The last section (Section 14.3) shows their combined output.

14.1. Server

```
1  #!/usr/bin/rexx
2  /* File:        05-server-clr.rexp
3   * Description: Creates a networking server which waits for connections on address 127.0.0.1
4   *              (localhost) and port 2015. Any received data is written to standard output.
5   *              Note: Any other program listening on this port will prevent the server from starting.
6   * Shows:       - Import of static classes
7   *              - Invokation of methods (with/without parameters)
8   *              - Instantiation of classes (with parameters)
9   *              - Retrieval of properties
10  */
```

The server part of the fifth example (see Listing E.5) demonstrates the usage of .NET classes to listen for an incoming TCP connection and receive data through it.

```
12 ipAddress = clr.import("System.Net.IPAddress")~Parse("127.0.0.1")  -- create "System.Net.IPAddress"
       object representing the localhost IP
13 tcpListener = .clr~new("System.Net.Sockets.TcpListener", ipAddress, 2015)  -- create instance of class
       "System.Net.Sockets.TcpListener"
```

[Lines 12-13] A new instance of `System.Net.IPAddress` is created by importing its static class and calling the .NET method `public static IPAddress Parse(string ipString)` with the IP address of localhost as parameter. It is then, along with the port number 2015, passed to the

constructor of `System.Net.Sockets.TcpListener` which creates an instance that is able to listen for incoming connections on the specified IP address and port.

```
15 SAY "Starting server..."
16 tcpListener~clr.dispatch("Start")  -- start the TCP listener
17 SAY "Waiting for connections..."
18 tcpSocket = tcpListener~AcceptSocket  -- wait for connections
19 SAY "Client connected."
```

[Line 16] Basically, this line calls the .NET method `public void Start()`, which starts the TCP listener. The use of `clr.dispatch` (see Section 9.2.3) is necessary, because `tcpListener` references a CLR object, which is an ooRexx class. It therefore automatically extends the ooRexx base class .Object, which defines its own method `start`. To prevent the default method from being called, `clr.dispatch` reroutes the call directly to the encapsulated .NET object.

[Line 18] The call of `public Socket AcceptSocket()` puts the program into a blocking state, i.e. execution will halt at this line until a connection is made.

```
21 buffer = clr.createArray("System.Byte", 1024)  -- create array of "System.Byte" with a size of 1MB)
22 tcpSocket~Receive(buffer)  -- write received data into "buffer"
```

[Line 21] This code will only be reached after a connection has been established. A buffer for the data to be received is created by letting `clr.createArray` (see Section 9.1.7) instantiate a .NET array of type `System.Byte`.

[Line 22] The .NET method `public int Receive(byte[] buffer)` will block code execution upon data is received. Once data is sent via the established TCP connection, it is written into the created array `buffer`.

```
24 SAY "Message received:"
25 decodedMessage = clr.import("System.Text.Encoding")~ASCII~GetString(buffer)  -- convert message from
       byte array to string
26 SAY decodedMessage
```

[Line 25] The received data is decoded by using the static .NET class `System.Text.Encoding`, which converts the received bytes into a textual representation.

```
28 tcpListener~Stop  -- stop TCP listener
29
30 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

[Lines 28-30] Finally, the opened TCP listener is stopped again to free up any used ports. As in any other example, ooRexx.NET is included using the `::REQUIRES` directive of ooRexx.

14.2. Client

```
1  #!/usr/bin/rexx
2  /* File:        05-client-clr.rexp
3   * Description: Creates a networking client which connects to a server listening on address 127.0.0.1
4   *              (localhost) and port 2015. User input is sent to the server.
5   *              Note: Sending a message to any other program than "05-server-clr.rexp" listening on
6   *              this port by chance might cause unpredictable behavior.
7   * Shows:       - Instantiation of classes (without parameters)
8   *              - Invokation of methods (with/without parameters)
9   *              - Retrieval of properties
10  */
```

The client part of the fifth example (see Listing E.6) demonstrates the usage of .NET classes to establish a TCP connection with a server and send data through it.

```
12 tcpClient = .clr~new("System.Net.Sockets.TcpClient")  -- create instance of class
       "System.Net.Sockets.TcpClient"
13
14 SAY "Connecting to 127.0.0.1:2015..."
15 tcpClient~Connect("127.0.0.1", 2015)  -- connect "TcpClient" to specified address and port
16 SAY "Connected."
```

[Lines 12-16] A new instance of `System.Net.Sockets.TcpClient` is instantiated and connected to a server listening on the specified IP address and port.

```
18 SAY "Input message to server:"
19 PARSE PULL message  -- wait for user input on console and store it in "message"
20 encodedMessage = clr.import("System.Text.Encoding")~ASCII~GetBytes(message)  -- convert input into
       byte array
21
22 SAY "Sending message..."
23 tcpClient~GetStream~Write(encodedMessage, 0, encodedMessage~Length)  -- send encoded message to server
```

[Line 19] The program waits for user input which it stores in `message` once entered.

[Line 20] The entered message is encoded by using the static .NET class `System.Text.Encoding`, which converts the message into an array of `System.Byte`.

[Line 23] The encoded message is sent by using the established TCP clients property `GetStream` (a `System.Net.Sockets.NetworkStream`) and its method `public override void Write(byte[] buffer, int offset, int size)`.

```
25 SAY "Message was sent to server."
26
27 tcpClient~Close  -- close TCP connection
28
29 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

[Lines 27-29] Finally, the opened TCP client is closed to free up any used ports. As in any other example, ooRexx.NET is included using the `::REQUIRES` directive of ooRexx.

14.3. Output

To execute the example, the server (see Section 14.1, console output shown in red) has to be started first. After it shows the following output, the client (see Section 14.2, console output shown in blue) can be started:

```
1 Starting server...
2 Waiting for connections...
```

After starting the client, the output of the two programs is continued as follows and the client is waiting for input:

```
3 Connecting to 127.0.0.1:2015...
4 Client connected.
5 Connected.
6 Input message to server:
```

By inputing a message in the client, e.g. ”This is a test message from the client.”, the output continues as shown below:

```
7 This is a test message from the client.
8 Sending message...
9 Message received:
10 This is a test message from the client.
11 Message was sent to server.
```

Part V.
Conclusion and Outlook

As the search for appropriate bridges between Java and .NET showed (see Section 5), the number of existing solutions is exiguous. Apart from very special use cases of companies (e.g. see [5] for customers), the possibility to use .NET from Java seems not to be in great demand.

The decision to use jni4net in ooRexx.NET was practically predetermined, as it was the only solution which was open source and supporting the much needed BSF4ooRexx project. However, during the implementation of ooRexx.NET a severe shortcoming was uncovered, as jni4net is not able to support generic programming at the moment, i.e. .NET classes or methods using generics. Therefore, certain classes can not be used with this version of ooRexx.NET.

Development of the library was conducted by first designing the practical examples (see Part IV) and implementing the functions needed to execute them afterwards. Consequently, all features shown by the examples are working, while code not available in any example might or might not work, though greatest care was taken upon implementation of the internal functions. Nevertheless, several features, e.g. the creation of .NET arrays with specified content, are certainly missing and provide one potential for future development.

Possibilities for enhancements and future development directions are:

- Enable support for generics
  If not done by a future version of jni4net, wrapping methods implemented on the Java and/or .NET side of the extended library could be used to camouflage generic functions while still making them available on the ooRexx side.
- Extend support for .NET arrays
  At the moment, only the creation of a .NET array without content is possible. This should be extended, so that additional parameters get added as array items.
- Clearer distinction between ooRexx.NET and BSF4ooRexx
  In the current version of the library, it is sometimes unclear whether a `CLR` or a `BSF_REFERENCE` is returned. For an end user, this should always be a `CLR`.

Appendix

A. References

[1] *About Open Object Rexx*. 2015. URL: http://www.oorexx.org/about.html (visited on 08/26/2015).

[2] Mike F. Cowlishaw. "The design of the REXX language". *IBM Systems Journal* 23(4) (1984), pp. 326–335.

[3] Rony G. Flatscher. *Introduction to Rexx and ooRexx*. Vienna: Facultas, 2013.

[4] *IKVM.NET Home Page*. 2015. URL: http://www.ikvm.net/ (visited on 08/26/2015).

[5] *javOnet - Java to .NET Bridge, C#, VB.NET*. 2015. URL: https://www.javonet.com/ (visited on 08/26/2015).

[6] *jni4net - bridge between Java and .NET*. 2015. URL: http://jni4net.com/ (visited on 08/26/2015).

[7] *Wikipedia: Common Intermediate Language*. 2015. URL: https://en.wikipedia.org/w/index.php?title=Common_Intermediate_Language&oldid=672235590 (visited on 08/26/2015).

[8] *Wikipedia: Common Language Infrastructure*. 2015. URL: https://en.wikipedia.org/w/index.php?title=Common_Language_Infrastructure&oldid=667836178 (visited on 08/26/2015).

[9] *Wikipedia: IKVM.NET*. 2015. URL: https://en.wikipedia.org/w/index.php?title=IKVM.NET&oldid=654784177 (visited on 08/26/2015).

[10] *Wikipedia: Java Class Library*. 2015. URL: https://en.wikipedia.org/w/index.php?title=Java_Class_Library&oldid=666317751 (visited on 08/26/2015).

[11] *Wikipedia: Java Native Interface*. 2015. URL: https://en.wikipedia.org/w/index.php?title=Java_Native_Interface&oldid=676092361 (visited on 08/26/2015).

[12] *Wikipedia: Java (programming language)*. 2015. URL: https://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=677906651 (visited on 08/26/2015).

[13] *Wikipedia: Java (software platform)*. 2015. URL: https://en.wikipedia.org/w/index.php?title=Java_(software_platform)&oldid=677903205 (visited on 08/26/2015).

[14] *Wikipedia: Java virtual machine*. 2015. URL: https://en.wikipedia.org/w/index.php?title=Java_virtual_machine&oldid=675920528 (visited on 08/26/2015).

[15] *Wikipedia: JavONet*. 2015. URL: https://en.wikipedia.org/w/index.php?title=JavOnet&oldid=611318654 (visited on 08/26/2015).

[16] *Wikipedia: .NET Framework*. 2015. URL: https://en.wikipedia.org/w/index.php?title=.NET_Framework&oldid=677752604 (visited on 08/26/2015).

B. Tables

C. Figures

D. Listings

E. Examples

```
#!/usr/bin/rexx
/* File:        01-helloworld-clr.rexp
 * Description: Simple output of "Hello World" via .NET classes.
 * Shows:       - Import of static classes
 *              - Invokation of methods (with parameters)
 */

console = clr.import("System.Console")  -- get reference to static class "System.Console"
console~WriteLine("Hello World from ooRexx.NET (via BSF4ooRexx and jni4net)")  -- invoke method "WriteLine" with parameter "Test"

::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

Listing E.1: ooRexx.NET - Example 1: Hello World (01-helloworld-clr.rxj)

```
#!/usr/bin/rexx
/* File:        02-eventlog-clr.rexp
 * Description: Reads the "Application" log of the current machine and outputs every entry. Uses the
 *              "System.Media.SystemSounds" class to emit a beep for every written message.
 * Shows:       - Import of static classes
 *              - Instantiation of classes (with parameters)
 *              - Retrieval of properties
 *              - Invokation of methods (without parameters)
 */

systemSounds = clr.import("System.Media.SystemSounds")  -- get reference to static class "System.Media.SystemSounds"

eventLog = .clr~new("System.Diagnostics.EventLog", "Application")  -- create instance of class "System.Diagnostics.EventLog" with parameter "Application"
logEvents = eventLog~Entries~GetEnumerator  -- retrieve log entry collection, create enumerator

DO WHILE logEvents~MoveNext <> .false  -- iterate over log entries
  eventEntry = logEvents~Current
  systemSounds~Beep~Play  -- play system sound "beep"
  SAY eventEntry~TimeGenerated "::" eventEntry~Message  -- output time and message of event
END

::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

Listing E.2: ooRexx.NET - Example 2: Event Log (02-eventlog-clr.rxj)

```
#!/usr/bin/rexx
/* File:        03-systemevents-clr.rexp
 * Description: Reacts to a Windows system event, namely a change of the systems clock.
 * Shows:       - Import of static classes
 *              - Instantiation of classes (without parameters)
 *              - Creation and registration of event handlers
 */

systemEvents = clr.import("Microsoft.Win32.SystemEvents")  -- get reference to static class "Microsoft.Win32.SystemEvents"

```

```
11 systemEventHandler = clr.createEventHandler(.SystemEventHandler~new)  -- create new event handler from
       ooRexx class "SystemEventHandler"
12 systemEvents~TimeChanged += systemEventHandler  -- register event handler to "TimeChanged" event
13
14 SAY "Waiting for ""TimeChanged"" event..."
15 PARSE PULL  -- wait for user input to prevent program from closing
16
17 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
18
19 /* Class:       SystemEventHandler
20  * Description: Implements the .NET event handler invoked by the "TimeChanged" event.
21  */
22 ::CLASS SystemEventHandler
23
24   /* Method:      invoke
25    * Description: Called by .NET when the registered event is triggered.
26    * Arguments:   - caller: the object the event handler was registered to
27    *              - eventArgs: the event arguments passed by the caller
28    */
29   ::METHOD invoke
30     USE ARG caller, eventArgs
31
32     SAY "Windows clock has been changed!"
```

Listing E.3: ooRexx.NET - Example 3: System Events (03-systemevents-clr.rxj)

```
1  #!/usr/bin/rexx
2  /* File:        04-forms-clr.rexp
3   * Description: Creates a windows form with a progress bar and a start button which, when clicked,
4   *              starts a new thread to increase the progress bar.
5   * Shows:       - Instantiation of classes (without parameters)
6   *              - Manipulation of properties
7   *              - Invokation of methods (with parameters)
8   *              - Creation and registration of event handlers
9   *              - Import of static classes
10  */
11
12 winForm = .clr~new("System.Windows.Forms.Form")  -- create instance of class
       "System.Windows.Forms.Form"
13 winForm~Text = "Processor"  -- set property "Text" to string "Processor"
14 winForm~AutoSize = .true  -- set property "AutoSize" to boolean true
15 winForm~AutoSizeMode = GrowAndShrink  -- set property "AutoSizeMode" to enumeration value
       "GrowAndShrink"
16
17 contentPane = .clr~new("System.Windows.Forms.FlowLayoutPanel")
18 contentPane~AutoSize = .true  -- set property "AutoSize" to boolean true
19 contentPane~AutoSizeMode = GrowAndShrink  -- set property "AutoSizeMode" to enumeration value
       "GrowAndShrink"
20 winForm~Controls~Add(contentPane)  -- add "FlowLayoutPanel" to "Form"
21
22 progressBar = .clr~new("System.Windows.Forms.ProgressBar")
23 progressBar~Minimum = 0  -- set property "Minimum" to integer 0
24 progressBar~Maximum = 100  -- set property "Maximum" to integer 100
25 progressBar~Value = 0  -- set property "Value" to integer 0
26 contentPane~Controls~Add(progressBar)  -- add "ProgressBar" to "FlowLayoutPanel"
27
```

```
28 startButton = .clr~new("System.Windows.Forms.Button")
29 startButton~Text = "Start"  -- set property "Text" to string "Start"
30 contentPane~Controls~Add(startButton)  -- add "Button" to "FlowLayoutPanel"
31
32 mouseEventHandler = clr.createEventHandler(.MouseEventHandler~new(progressBar, startButton))  --
       create new event handler from ooRexx class "MouseEventHandler"
33 startButton~Click += mouseEventHandler  -- register event handler to "Click" event
34
35 application = clr.import("System.Windows.Forms.Application")  -- get reference to static class
       "System.Windows.Forms.Application"
36 application~Run(winForm)  -- invoke method "Run", which starts an application message loop
37
38 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
39
40 /* Class:       MouseEventHandler
41  * Description: Implements the .NET event handler invoked by the "Click" event.
42  */
43 ::CLASS MouseEventHandler
44
45   /* Method:      init
46    * Description: Constructor which saves the passed arguments in the class.
47    * Arguments:   - progressBar: the progress bar which is to be modified
48    *              - startButton: the start button which is to be modified
49    */
50   ::METHOD init
51     EXPOSE progressBar startButton
52     USE ARG progressBar, startButton
53
54   /* Method:      invoke
55    * Description: Called by .NET when the registered event is triggered.
56    * Arguments:   - caller: the object the event handler was registered to
57    *              - mouseEventArgs: the event arguments passed by the caller
58    */
59   ::METHOD invoke
60     EXPOSE progressBar startButton
61     USE ARG caller, mouseEventArgs
62
63     .Processor~new(progressBar, startButton)~start  -- creates a new instance of ooRexx class
         "Processor", which is a thread, and starts it
64
65 /* Class:       Processor
66  * Description: Inherits from CLRThread defined in CLR.CLS, which enables this class to run in its own
67  *              thread. The "run" method has to be implemented, as it is the method executed upon
68  *              start of the thread. The thread must be started by invoking "start", which is a method
69  *              implemented in CLRThread. Invoking "run" will not start a concurrent execution of the
70  *              method!
71  */
72 ::CLASS Processor SUBCLASS CLRThread
73
74   /* Method:      init
75    * Description: Constructor which saves the passed arguments in the class.
76    * Arguments:   - progressBar: the progress bar which is to be modified
77    *              - startButton: the start button which is to be modified
78    */
79   ::METHOD init
80     EXPOSE progressBar startButton
```

```
81      USE ARG progressBar, startButton
82
83    /* Method:      run
84     * Description: Executed upon start of the thread (by invoking "start").
85     */
86    ::METHOD run
87      EXPOSE progressBar startButton
88
89      startButton~Enabled = .false  -- disable start button to prevent multiple clicks
90
91      DO i = 1 TO 100
92        progressBar~Value = i  -- set value of the progress bar (from 1 to 100)
93        CALL syssleep .1  -- sleep 100 milliseconds to prevent reaching 100 immediately
94      END
95
96      startButton~Text = "Finished"  -- set text on button to "Finished"
```

Listing E.4: ooRexx.NET - Example 4: Forms (04-forms-clr.rxj)

```
1  #!/usr/bin/rexx
2  /* File:        05-client-clr.rexp
3   * Description: Creates a networking client which connects to a server listening on address 127.0.0.1
4   *              (localhost) and port 2015. User input is sent to the server.
5   *              Note: Sending a message to any other program than "05-server-clr.rexp" listening on
6   *              this port by chance might cause unpredictable behavior.
7   * Shows:       - Instantiation of classes (without parameters)
8   *              - Invokation of methods (with/without parameters)
9   *              - Retrieval of properties
10  */
11
12 tcpClient = .clr~new("System.Net.Sockets.TcpClient")  -- create instance of class
       "System.Net.Sockets.TcpClient"
13
14 SAY "Connecting to 127.0.0.1:2015..."
15 tcpClient~Connect("127.0.0.1", 2015)  -- connect "TcpClient" to specified address and port
16 SAY "Connected."
17
18 SAY "Input message to server:"
19 PARSE PULL message  -- wait for user input on console and store it in "message"
20 encodedMessage = clr.import("System.Text.Encoding")~ASCII~GetBytes(message)  -- convert input into
       byte array
21
22 SAY "Sending message..."
23 tcpClient~GetStream~Write(encodedMessage, 0, encodedMessage~Length)  -- send encoded message to server


24
25 SAY "Message was sent to server."
26
27 tcpClient~Close  -- close TCP connection
28
29 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

Listing E.5: ooRexx.NET - Example 5: Client (05-client-clr.rxj)

```
1  #!/usr/bin/rexx
2  /* File:        05-server-clr.rexp
```

```
3  * Description: Creates a networking server which waits for connections on address 127.0.0.1
4  *              (localhost) and port 2015. Any received data is written to standard output.
5  *              Note: Any other program listening on this port will prevent the server from starting.
6  * Shows:       - Import of static classes
7  *              - Invokation of methods (with/without parameters)
8  *              - Instantiation of classes (with parameters)
9  *              - Retrieval of properties
10 */
11
12 ipAddress = clr.import("System.Net.IPAddress")~Parse("127.0.0.1")  -- create "System.Net.IPAddress"
       object representing the localhost IP
13 tcpListener = .clr~new("System.Net.Sockets.TcpListener", ipAddress, 2015)  -- create instance of class
       "System.Net.Sockets.TcpListener"
14
15 SAY "Starting server..."
16 tcpListener~clr.dispatch("Start")  -- start the TCP listener
17 SAY "Waiting for connections..."
18 tcpSocket = tcpListener~AcceptSocket  -- wait for connections
19 SAY "Client connected."
20
21 buffer = clr.createArray("System.Byte", 1024)  -- create array of "System.Byte" with a size of 1MB)
22 tcpSocket~Receive(buffer)  -- write received data into "buffer"
23
24 SAY "Message received:"
25 decodedMessage = clr.import("System.Text.Encoding")~ASCII~GetString(buffer)  -- convert message from
       byte array to string
26 SAY decodedMessage
27
28 tcpListener~Stop  -- stop TCP listener
29
30 ::REQUIRES CLR.CLS  -- get ooRexx.NET support
```

Listing E.6: ooRexx.NET - Example 5: Server (05-server-clr.rxj)

F. CLR.CLS

```
#!/usr/bin/rexx
/* File:        CLR.CLS
 * Description: Library for making Microsoft's .NET Framework available in ooRexx: .NET classes and
 *              objects appear as ooRexx classes and objects and one can send ooRexx messages to them.
 *              Requires BSF4ooRexx and jni4net.
 */

.local~clr.bridge = BSF.import("net.sf.jni4net.Bridge")  -- get jni4net support
.clr.bridge~setVerbose(.false)  -- .true for debug output

.clr.bridge~bsf.dispatch("init")  -- initialize jni4net support

.local~clr.assembly = BSF.import("system.reflection.Assembly")

ooRexxAssembly = .clr.assembly~LoadWithPartialName("oorexx.net")  -- get additional .NET proxies
.clr.bridge~RegisterAssembly(ooRexxAssembly)

CALL clr.initAssemblies

.local~clr.system.array = BSF.import("system.Array")  -- preload frequently used classes
.local~clr.system.enum = BSF.import("system.Enum")
.local~clr.system.object = BSF.import("system.Object")
.local~clr.system.type = BSF.import("system.Type")

.CLRLogger~setLevel("OFF")  -- turn off logging by default ("OFF")

::REQUIRES BSF.CLS  -- get BSF4ooRexx support

/* Routine:     clr.initAssemblies
 * Description: Stores class names defined in commonly used .NET assemblies ("mscorlib" and "System")
 *              in the local directory .clr.assemblyName for fast and easy lookup.
 */
::ROUTINE clr.initAssemblies PRIVATE

  .local~clr.assemblyName = .directory~new

  CALL clr.addAssembly "mscorlib"
  CALL clr.addAssembly "System"

/* Routine:     clr.addAssembly
 * Description: Retrieves class names defined in given assembly name and adds them to the local
 *              directory .clr.assemblyName for fast and easy lookup.
 * Arguments:   - assemblyName: the assembly name which is to be searched for classes
 */
::ROUTINE clr.addAssembly PUBLIC
  PARSE ARG assemblyName

  assemblyTypes = .clr.assembly~LoadWithPartialName(assemblyName)~GetExportedTypes  -- retrieve
      classes defined in assembly with given name

  DO i = 1 TO assemblyTypes~size
    .clr.assemblyName~setEntry(assemblyTypes[i]~getFullName, assemblyName)
  END

```

```
54 /* Routine:     clr.extractAssemblyName
55  * Description: Tries to determine the assembly name of the given class by splitting it and removing
56  *              the last part. Only works for classes where the fully qualified name is just one
57  *              "level" above its assembly name. If the class is in a different assembly, it needs to
58  *              be added to the known assemblies by calling clr.addAssembly.
59  * Arguments:   - className: the (fully qualified) class name of a .NET class
60  * Returns:     the (supposed) assembly name
61  */
62 ::ROUTINE clr.extractAssemblyName PRIVATE
63   PARSE ARG className
64
65   split = LASTPOS(".", className)
66   assemblyName = SUBSTR(className, 1, (split-1))  -- extract assembly name from fully qualified class
         name
67
68   RETURN assemblyName
69
70 /* Routine:     clr.findAssemblyName
71  * Description: Determines the assembly name for the given (fully qualified) class name. If the class
72  *              can not be found in the known assemblies (as instantiated by clr.initAssemblies), the
73  *              (supposed) assembly name is returned as determined by clr.extractAssemblyName.
74  * Arguments:   - className: the (fully qualified) class name of a .NET class
75  * Returns:     the (probably supposed) assembly name
76  */
77 ::ROUTINE clr.findAssemblyName PRIVATE
78   PARSE ARG className
79
80   assemblyName = .clr.assemblyName~entry(className)  -- lookup assembly name in known assemblies
81
82   IF assemblyName <> .nil THEN
83     RETURN assemblyName
84   ELSE
85     RETURN clr.extractAssemblyName(className)
86
87 /* Routine:     clr.import
88  * Description: Creates and returns an instance of CLRClass which provides a reference to a static
89  *              .NET class.
90  * Arguments:   - className: the (fully qualified) class name of the static .NET class
91  * Returns:     a CLRClass referencing the given .NET class
92  */
93 ::ROUTINE clr.import PUBLIC
94   PARSE ARG className
95
96   clrClass = .CLRClass~new(className)
97
98   RETURN clrClass
99
100 /* Routine:     clr.createEventHandler
101  * Description: Makes given ooRexx class inherit from .NET class "System.EventHandler" to enable it to
102  *              receive events triggered by .NET objects.
103  * Arguments:   - rexxEventHandler: the ooRexx class which handles the event
104  *              - rexxData: optional argument to be included in the proxies slotDir~userdata value
105  * Returns:     a Java proxy to the given ooRexx class
106  */
107 ::ROUTINE clr.createEventHandler PUBLIC
108   USE ARG rexxEventHandler, rexxData = .nil
```

```
109
110   prxEventHandler = BSFCreateRexxProxy(rexxEventHandler, rexxData)
111   prxClass = bsf.createProxyClass("system.EventHandler")
112   eventHandler = prxClass~new(prxEventHandler)
113
114   RETURN eventHandler
115
116 /* Routine:     clr.createArray
117  * Description: Creates and returns a .NET array of given class and capacity wrapped in a Java proxy.
118  * Arguments:   - className: the type of the objects to be stored in the array
119  *              - capacity: the capacity of the array
120  * Returns:     a .NET array of the given class wrapped in a Java proxy
121  */
122 ::ROUTINE clr.createArray PUBLIC
123   PARSE ARG className, capacity
124
125   RETURN .clr.system.array~CreateInstance(.clr.system.type~GetType(className), capacity)
126
127 /* Routine:     clr.wrap
128  * Description: Processes received parameter "param" depending on its type. Always returns a CLR.
129  * Arguments:   - param: any string/object/instance/other which is supposed to be (in) a CLR
130  * Returns:     a CLR instance wrapping "param"
131  */
132 ::ROUTINE clr.wrap PRIVATE
133   USE ARG param
134
135   IF param~isA(.CLR) THEN  -- if "param" already is a CLR, nothing is to be done
136   DO
137     .CLRLogger~trace("clr.wrap: returning unchanged CLR")
138     RETURN param
139   END
140   ELSE IF param~isA(.string) THEN  -- if "param" is a string...
141   DO
142     IF VERIFY(param, "0123456789") = 0 THEN  -- return "System.Int32" if it contains only digits
143     DO
144       .CLRLogger~trace("clr.wrap: new System.Int32 CLR")
145       RETURN .clr~new("System.Int32", param)
146     END
147     ELSE  -- return "System.String" elsewise
148     DO
149       .CLRLogger~trace("clr.wrap: new System.String CLR")
150       RETURN .clr~new("System.String", param)
151     END
152   END
153   ELSE  -- return new CLR, let constructor of CLR handle "param"
154   DO
155     .CLRLogger~trace("clr.wrap: returning new CLR")
156     RETURN .clr~new(param)
157   END
158
159 /* Routine:     clr.findMatchingMethod
160  * Description: Searches given "Type" object for given method name.
161  * Arguments:   - clrType: a .NET "System.Type" defining its available methods
162  *              - methodName: the name of the method to be searched for
163  * Returns:     case sensitive name of the method or .nil if it could not be found
164  */
```

```
::ROUTINE clr.findMatchingMethod PRIVATE
  USE ARG clrType, methodName

  .CLRLogger~trace("Searching method name for" methodName "in" clrType)

  typeMethods = clrType~GetMethods  -- retrieve methods available for this "Type"
  typeMethodsCount = typeMethods~size

  DO i = 1 TO typeMethodsCount
    IF typeMethods[i]~getName~caselessEquals(methodName) THEN  -- case insensitive comparison
      RETURN typeMethods[i]~getName
  END

  RETURN .nil  -- return .nil only if no matching method was found

/* Routine:     clr.findMatchingProperty
 * Description: Searches given "Type" object for given property name.
 * Arguments:   - clrType: a .NET "System.Type" defining its available properties
 *              - propertyName: the name of the property to be searched for
 * Returns:     case sensitive name of the property or .nil if it could not be found
 */
::ROUTINE clr.findMatchingProperty PRIVATE
  USE ARG clrType, propertyName

  .CLRLogger~trace("Searching property name for" propertyName "in" clrType)

  typeProperties = clrType~GetProperties  -- retrieve properties available for this "Type"
  typePropertiesCount = typeProperties~size

  DO i = 1 TO typePropertiesCount
    IF typeProperties[i]~getName~caselessEquals(propertyName) THEN  -- case insensitive comparison
      RETURN typeProperties[i]~getName
  END

  RETURN .nil  -- return .nil only if no matching property was found

/* Routine:     clr.findMatchingEvent
 * Description: Searches given "Type" object for given event name.
 * Arguments:   - clrType: a .NET "System.Type" defining its available events
 *              - eventName: the name of the event to be searched for
 * Returns:     case sensitive name of the event or .nil if it could not be found
 */
::ROUTINE clr.findMatchingEvent PRIVATE
  USE ARG clrType, eventName

  .CLRLogger~trace("Searching event name for" eventName "in" clrType)

  typeEvents = clrType~GetEvents  -- retrieve events available for this "Type"
  typeEventsCount = typeEvents~size

  DO i = 1 TO typeEventsCount
    IF typeEvents[i]~getName~caselessEquals(eventName) THEN  -- case insensitive comparison
      RETURN typeEvents[i]~getName
  END

  RETURN .nil  -- return .nil only if no matching event was found
```

```
221
222 /* Class:       CLR
223  * Description: Wraps a .NET proxy and handles initialization and method calls to it.
224  */
225 ::CLASS CLR PUBLIC
226
227   /* Method:      init
228    * Description: Constructor which creates an instance of CLR based on the supplied class name and
229    *              parameters.
230    * Arguments:   - className: name of the .NET class to be instantiated or a BSF proxy to be wrapped
231    *              - param: first parameter for the class, default .nil
232    *              - ...: additional parameters
233    * Returns:     the freshly created CLR
234    */
235   ::METHOD init PUBLIC
236     EXPOSE clrInstance clrType
237     USE ARG className, param = .nil, ...
238
239     .CLRLogger~trace("Creating new CLR instance of" className "with parameters" param)
240
241     IF className~isA(.string) THEN
242     DO
243       assemblyName = clr.findAssemblyName(className)  -- find assembly of (fully qualified) class name
244
245       SELECT
246         WHEN className = "System.Boolean" THEN  -- special handling for boolean parameters and
              instantiation
247         DO
248           IF param = .true | param~caselessEquals("true") THEN param = "true"
249           ELSE param = "false"
250
251           clrInstance = .clr.assembly~LoadWithPartialName(assemblyName)~CreateInstance(className)
252           clrInstance = clrInstance~GetType~GetMethod("Parse", bsf.createJavaArrayOf(.clr.system.type,
                  .clr.system.type~GetType("System.String")))~Invoke(clrInstance,
                bsf.createJavaArrayOf(.clr.system.object, .clr.bridge~convert(param)))
253         END
254
255         WHEN className = "System.String" THEN  -- special handling for string parameters and
              instantiation
256         DO
257           clrInstance = .bsf~new("system.String", param)
258         END
259
260         WHEN className = "System.Int32" THEN  -- special handling for integer parameters and
              instantiation
261         DO
262           clrInstance = .clr.assembly~LoadWithPartialName(assemblyName)~CreateInstance(className)
263           clrType = clrInstance~GetType
264           clrInstance = clrType~GetMethod("Parse", bsf.createJavaArrayOf(.clr.system.type,
                  .clr.system.type~GetType("System.String")))~Invoke(clrInstance,
                bsf.createJavaArrayOf(.clr.system.object, .clr.bridge~convert(param)))
265         END
266
267         OTHERWISE  -- all other classes can be done in an unified way
268         DO
269           argCount = arg() - 1
```

```
270
271           IF argCount = 0 THEN  -- no arguments, can use default constructor
272             clrInstance = .clr.assembly~LoadWithPartialName(assemblyName)~CreateInstance(className)
273           ELSE  -- arguments need to be wrapped in an array to be passed to appropriate constructor
274           DO
275             argsList = bsf.createJavaArray(.clr.system.object, argCount)
276
277             DO i = 1 TO argCount
278               .CLRLogger~trace("Parsing argument" arg(i+1))
279               argument = clr.wrap(arg(i+1))
280               .CLRLogger~trace("Parsed argument" argument argument~clr.getType
                      argument~clr.getInstance)
281               argsList[i] = argument~clr.getInstance
282             END
283
284             clrInstance = .clr.assembly~LoadWithPartialName(assemblyName)~CreateInstance(className,
                    .false, .nil, .nil, argsList, .nil, .nil)
285           END
286         END
287       END
288
289       clrType = clrInstance~GetType
290     END
291     ELSE  -- supplied argument is (supposed to be) a BSF_REFERENCE
292     DO
293       clrInstance = className
294       clrType = className~GetType
295     END
296
297     .CLRLogger~trace("Created CLR instance of" className "with clrInstance" clrInstance "and clrType"
          clrType)
298
299   /* Method:      unknown
300    * Description: Catches every message sent to this CLR and determines how it is to be treated.
301    * Arguments:   - command: the message sent to this CLR
302    *              - args: the arguments of the message
303    * Returns:     depending on the command
304    */
305   ::METHOD unknown PUBLIC
306     EXPOSE clrInstance clrType
307     USE ARG command, args
308
309     IF RIGHT(command,1) = "=" THEN  -- check if it is an assignment
310     DO
311       IF args[1]~isA(.CLREvent) THEN  -- check passed argument, might already be handled ("+=")
312         RETURN args[1]
313
314       PARSE VAR command property "=" .  -- extract property name (without "=")
315
316       propertyName = clr.findMatchingProperty(clrType, property)  -- try to find given property in
              "Type"
317
318       IF args~items = 1 THEN
319         RETURN self~clr.setPropertyValue(propertyName, args[1])  -- set property value to given
                argument
320     END
```

```
321     ELSE  -- not an assignment
322     DO
323       methodName = clr.findMatchingMethod(clrType, command)  -- try to find given method in "Type"
324
325       IF methodName <> .nil THEN  -- aimed for a method
326       DO
327         typeList = bsf.createJavaArray(.clr.system.type, args~items)
328         argsList = bsf.createJavaArray(.clr.system.object, args~items)
329
330         IF args~items > 0 THEN  -- arguments need to be wrapped in an array to be passed to method
331         DO i = 1 TO args~items
332           .CLRLogger~trace("Parsing argument" args[i]~class)
333           argument = clr.wrap(args[i])
334           .CLRLogger~trace("Parsed argument" argument argument~clr.getType argument~clr.getInstance)
335           typeList[i] = argument~clr.getType
336           argsList[i] = argument~clr.getInstance
337         END
338
339         result = clrInstance~GetType~GetMethod(methodName, typeList)~Invoke(clrInstance, argsList)  --
                invoke given method with supplied parameters
340
341         IF result <> .nil THEN
342           RETURN .clr~new(result)
343         ELSE
344           RETURN .nil
345       END
346       ELSE  -- aimed for another member
347       DO
348         propertyName = clr.findMatchingProperty(clrType, command)  -- try to find given property in
                "Type"
349
350         IF propertyName <> .nil THEN  -- aimed for a property
351         DO
352           propertyInfo = clrInstance~GetType~GetProperty(propertyName)
353
354           RETURN .clr~new(propertyInfo~GetValue(clrInstance, .nil))  -- return value of property
355         END
356         ELSE  -- aimed for an event
357         DO
358           eventName = clr.findMatchingEvent(clrType, command)  -- try to find given event in "Type"
359
360           RETURN .CLREvent~new(self, eventName)  -- return new CLREvent
361         END
362       END
363     END
364
365   /* Method:      clr.dispatch
366    * Description: Directly forwards a message to the "unknown" method of this CLR. This is needed if a
367    *              method on .NET side is to be called which is named like a method of this CLR object.
368    *              Examples are "start" or "init" (inherited from .Object).
369    * Arguments:   - methodName: name of the method to be invoked
370    * Returns:     the return value of the unknown method
371    */
372   ::METHOD clr.dispatch PUBLIC
373     PARSE ARG methodName
374
```

```
375     RETURN self~unknown(methodName, arg(2, 'A'))  -- calls unknown method
376
377   /* Method:      string
378    * Description: Overrides "string" method of .Object to enable certain CLR instances to output their
379    *              actual content instead of their class name.
380    * Returns:     the value of the contained proxy
381    */
382   ::METHOD string PUBLIC
383     RETURN self~clr.getInstance~toString
384
385   /* Method:      clr.getType
386    * Description: Returns the "System.Type" of the .NET object wrapped in this CLR.
387    * Returns:     the "Type" of the .NET object in this CLR
388    */
389   ::METHOD clr.getType PUBLIC
390     EXPOSE clrType
391     RETURN clrType
392
393   /* Method:      clr.getInstance
394    * Description: Returns the actual instance of the .NET object wrapped in this CLR.
395    * Returns:     the actual instance of the .NET object in this CLR
396    */
397   ::METHOD clr.getInstance PUBLIC
398     EXPOSE clrInstance
399     RETURN clrInstance
400
401   /* Method:      clr.setPropertyValue
402    * Description: Sets the specified property to the supplied value in the .NET object wrapped in
403    *              this CLR.
404    * Arguments:   - propertyName: the name of the property to be set
405    *              - propertyValue: the value the property is to be set to
406    * Returns:     self
407    */
408   ::METHOD clr.setPropertyValue PRIVATE
409     EXPOSE clrInstance clrType
410     USE ARG propertyName, propertyValue
411
412     .CLRLogger~trace("Setting property" propertyName "to" propertyValue)
413
414     IF propertyValue~isA(.CLR) THEN  -- if supplied value is a CLR, extract its wrapped instance
415       propertyValue = propertyValue~clr.getInstance
416     ELSE  -- else handle value according to its (expected) type
417     DO
418       propertyType = clrType~GetProperty(propertyName)~GetPropertyType() -- get expected datatype for
              this property
419
420       IF propertyType~isEnum = .true THEN  -- special handling for enums
421         propertyValue = .clr.system.enum~Parse(propertyType, propertyValue, .true)
422       ELSE
423         propertyValue = .clr~new(propertyType~toString, propertyValue)~clr.getInstance
424     END
425
426     clrType~GetProperty(propertyName)~SetValue(clrInstance, propertyValue, .nil)  -- actually set
          value
427
428     RETURN self
```

```
429
430 /* Class:       CLRClass
431  * Description: Wraps a .NET proxy of a static class and handles its initialization and method calls.
432  */
433 ::CLASS CLRClass PRIVATE
434
435   /* Method:      init
436    * Description: Constructor which creates an instance of CLRClass based on the supplied class name.
437    * Arguments:   - className: name of the static .NET class to be referenced
438    * Returns:     the freshly created CLRClass
439    */
440   ::METHOD init PUBLIC
441     EXPOSE clrClass clrType
442     USE ARG className
443
444     .CLRLogger~trace("Creating new CLRClass instance of" className)
445
446     assemblyName = clr.findAssemblyName(className)  -- find assembly of (fully qualified) class name
447
448     clrClass = .clr.assembly~LoadWithPartialName(assemblyName)
449     clrType = clrClass~GetType(className)
450
451     .CLRLogger~trace("Created CLRClass instance of" className "with clrClass" clrClass "and clrType"
          clrType)
452
453   /* Method:      unknown
454    * Description: Catches every message sent to this CLRClass and determines how it is to be treated.
455    * Arguments:   - command: the message sent to this CLR
456    *              - args: the arguments of the message
457    * Returns:     depending on the command
458    */
459   ::METHOD unknown PUBLIC
460     EXPOSE clrClass clrType
461     USE ARG command, args
462
463     IF RIGHT(command,1) = "=" THEN  -- check if it is an assignment
464     DO
465       IF args[1]~isA(.CLREvent) THEN  -- check passed argument, might already be handled ("+=")
466         RETURN args[1]
467
468       PARSE VAR command property "=" .  -- extract property name (without "=")
469
470       propertyName = clr.findMatchingProperty(clrType, property)  -- try to find given property in
            "Type"
471
472       IF args~items = 1 THEN
473         RETURN self~clr.setPropertyValue(propertyName, args[1])  -- set property value to given
              argument
474     END
475     ELSE
476     DO
477       methodName = clr.findMatchingMethod(clrType, command)  -- try to find given method in "Type"
478
479       IF methodName <> .nil THEN  -- aimed for a method
480       DO
481         typeList = bsf.createJavaArray(.clr.system.type, args~items)
```

```
482         argsList = bsf.createJavaArray(.clr.system.object, args~items)
483
484         IF args~items > 0 THEN  -- arguments need to be wrapped in an array to be passed to method
485         DO i = 1 TO args~items
486           .CLRLogger~trace("Parsing argument" args[i]~class)
487           argument = clr.wrap(args[i])
488           .CLRLogger~trace("Parsed argument" argument argument~clr.getType argument~clr.getInstance)
489           typeList[i] = argument~clr.getType
490           argsList[i] = argument~clr.getInstance
491         END
492
493         result = clrType~GetMethod(methodName, typeList)~Invoke(.nil, argsList)  -- invoke given
                method with supplied parameters
494
495         IF result <> .nil THEN
496           RETURN .clr~new(result)
497         ELSE
498           RETURN .nil
499       END
500       ELSE  -- aimed for another member
501       DO
502         propertyName = clr.findMatchingProperty(clrType, command)  -- try to find given property in
                "Type"
503
504         IF propertyName <> .nil THEN  -- aimed for a property
505         DO
506           propertyInfo = clrType~GetProperty(propertyName)
507
508           RETURN .clr~new(propertyInfo~GetValue(clrType, .nil))  -- return value of property
509         END
510         ELSE  -- aimed for an event
511         DO
512           eventName = clr.findMatchingEvent(clrType, command)  -- try to find given event in "Type"
513
514           RETURN .CLREvent~new(self, eventName)  -- return new CLREvent
515         END
516       END
517     END
518
519 /* Class:       CLREvent
520  * Description: Handles assignment and deassignment of event handlers.
521  */
522 ::CLASS CLREvent PRIVATE
523
524   /* Method:      init
525    * Description: Constructor which saves the passed arguments in the class.
526    * Arguments:   - clr: the CLR which wraps the object the event is registered to
527    *              - eventName: the name of the event
528    */
529   ::METHOD init PUBLIC
530     EXPOSE clr eventName
531     USE ARG clr, eventName
532
533   /* Method:      "+"
534    * Description: Is called upon assignment of an event handler with "+=".
535    * Arguments:   - eventHandler: the event handler created by clr.createEventHandler
```

```
536     * Returns:     self
537     */
538   ::METHOD "+" PUBLIC
539     EXPOSE clr eventName
540     USE ARG eventHandler
541
542     .CLRLogger~trace("Adding EventHandler " eventHandler "to" clr "(Event:" eventName ")")
543
544     addMethod = "add_" || eventName  -- jni4net reflects the method to add an event handler with
            "add_" and the event name
545     INTERPRET "clr~" || addMethod || "(eventHandler)"  -- let ooRexx carry out the statement in the
            string
546
547     RETURN self
548
549   /* Method:      "-"
550    * Description: Is called upon deassignment of an event handler with "-=".
551    * Arguments:   - eventHandler: the event handler created by clr.createEventHandler
552    * Returns:     self
553    */
554   ::METHOD "-" PUBLIC
555     EXPOSE clr eventName
556     USE ARG eventHandler
557
558     .CLRLogger~trace("Removing EventHandler " eventHandler "from" clr "(Event:" eventName ")")
559
560     removeMethod = "remove_" || eventName  -- jni4net reflects the method to remove an event handler
            with "remove_" and the event name
561     INTERPRET "clr~" || removeMethod || "(eventHandler)"  -- let ooRexx carry out the statement in the
            string
562
563     RETURN self
564
565 /* Class:       CLRThread
566  * Description: Provides the environment to start threads interacting with .NET objects. To use it,
567  *              the ooRexx class needs to subclass CLRThread and override the "run" method. It must
568  *              then be started with the "start" method to actually start a new thread.
569  */
570 ::CLASS CLRThread PUBLIC
571
572   /* Method:      start
573    * Description: Creates a new thread which is able to flawlessly interact with .NET objects in
574    *              different threads. Must be called to execute the "run" method in a new thread.
575    * Arguments:   - rexxData: optional argument to be included in the proxies slotDir~userdata value
576    */
577   ::METHOD start PUBLIC
578     USE ARG rexxData = .nil
579     rexxRunnableProxy = BsfCreateRexxProxy(self, rexxData, "java.lang.Runnable")
580     rexxThread = .bsf~new("java.lang.Thread", rexxRunnableProxy)
581     rexxThread~bsf.dispatch("start")
582
583   /* Method:      run
584    * Description: Needs to be overridden by specialising ooRexx class.
585    */
586   ::METHOD run ABSTRACT
587
```

```
588 /* Class:       CLRLogger
589  * Description: Provides simple logging functionality within CLR.CLS and for programs using it.
590  *              Different log levels are defined and can be used to output all kinds of information
591  *              at runtime or for debugging purposes. The kind of messages displayed can be limited
592  *              by specifying the intended log level.
593  */
594 ::CLASS CLRLogger PUBLIC
595
596   /* Method:      init
597    * Description: Constructor which defines the available log levels.
598    */
599   ::METHOD init PUBLIC CLASS
600     EXPOSE logLevels
601
602     logLevels = .directory~new
603
604     logLevels["OFF"] = 70
605     logLevels["FATAL"] = 60
606     logLevels["ERROR"] = 50
607     logLevels["WARN"] = 40
608     logLevels["INFO"] = 30
609     logLevels["DEBUG"] = 20
610     logLevels["TRACE"] = 10
611
612   /* Method:      setLevel
613    * Description: Sets the log level limit. Only messages of this type (and above) will be outputted.
614    *              For example, a log level limit of "ERROR" will only output messages of type
615    *              "ERROR" and "FATAL".
616    * Arguments:   - desiredLevel: the log level limit
617    */
618   ::METHOD setLevel PUBLIC CLASS
619     EXPOSE logLevel logLevels
620     PARSE ARG desiredLevel
621
622     IF logLevels[desiredLevel] <> .nil THEN  -- check if supplied argument is a valid log level
623       logLevel = logLevels[desiredLevel]
624     ELSE
625       logLevel = logLevels["OFF"]
626
627   /* Method:      unknown
628    * Description: Catches log messages. For example, a log output of type "INFO" can be invoked by
629    *              .CLRLogger~info("log output"). Output is given (or not) depending on the set level
630    *              limit.
631    * Arguments:   - levelName: the log level this message belongs to
632    *              - args: the message itself
633    */
634   ::METHOD unknown PUBLIC CLASS
635     EXPOSE logLevel logLevels
636     USE ARG levelName, args
637
638     IF args~ITEMS > 0 & logLevels[levelName] <> .nil THEN  -- check if there is a message and if the
          given log level is defined
639     DO
640       message = args[1]
641
```

```
642       IF logLevels[levelName] >= logLevel THEN  -- only process message if invoked log level is
              greater or equal log level limit
643         .CLRLogger~output(levelName, message)
644     END
645
646   /* Method:      output
647    * Description: Provides formatting and actual output for a log message.
648    * Arguments:   - logLevel: the log level this message belongs to
649    *              - message: the message itself
650    */
651   ::METHOD output PRIVATE CLASS
652     PARSE ARG logLevel, message
653
654     SAY "[" || .dateTime~new || " | " ||  LEFT(logLevel, 5) || "] :: " || message  -- default output
          with date, time, log level (always five characters for consistent columns) and message
```

Listing F.1: ooRexx.NET: CLR.CLS