How to generate a LinkableComponent with a Fortran engine on Linux

Table of contents

1. Introduction
2. Technical prerequisites
   • 2.1. Machine and OS of the test system
   • 2.2. Mono for multi platform C#
   • 2.3. Fortran 90 Compiler
3. How to generate a shared library for a Fortran engine
   • 3.1. General
   • 3.2. Ifort parameter bug
   • 3.3. Fortran interface functions
4. How to port a C# wrapper from Windows to Linux
   • 4.1. General
   • 4.2. How to port the individual layers
     • 4.2.1. <engine>DllAccess.cs
     • 4.2.2. <engine>DotNetAccess.cs
     • 4.2.3. <engine>Wrapper.cs
5. Use of the LinkableComponent

1. Introduction

The document describes how to generate a LinkableComponent on Linux:

• how to compile a shared library for a Fortran 90 engine named <engine>
• how to port a C# wrapper for this shared library from Windows to Linux. The wrapper contains a class for accessing the Fortran dll <engine>DllAccess and two outer classes <engine>DotNetAccess and <engine>Wrapper, see Migrating existing Fortran based models codes

2. Technical prerequisites

2.1. Machine and OS of the test system

• workstation with an Intel Xeon processor
• 64bit Suse Linux Edition Desktop SLED 10.3 on the machine with the Fortran compiler
• 64bit openSUSE 11.0 on the machine with the C# compiler

2.2. Mono for multi platform C#

• Mono v. 1.9.1. for openSUSE 11.0 in 64 bit mode. V. 1.9.1. is the concrete term, but it is also referred to as Mono 2.0.
• compiler gmes 2.0.

2.3. Fortran 90 Compiler

• Intel 64 bit Fortran Compiler v. 9.1.051

3. How to generate a shared library for a Fortran engine

3.1. General

During runtime the C# wrapper refers only one Fortran shared library. All Fortran sources were compiled to one shared library <fortranEngine>.so:
ifort -c -fPIC -convert big_endian -fpp *.f90
- c : compile to object (.o) only, do not link
- fPIC : generate position independent code (for shared libs)
- convert big_endian: the order in which a sequence of bytes is stored in a computer’s memory,
  here: most significant bytes first;
- used, e.g. on mainframes and supercomputers.
- fpp : run Fortran preprocessor on source files prior to compilation

Linking the compiled objects:

ifort -shared -o <fortranEngine.so> *.o

During runtime some fortran libraries must be accessible via the environment variable LD_LIBRARY_PATH. In the following examples they are provided by the ifort compiler.

export LD_LIBRARY_PATH=/opt/intel/fce/9.1.051/lib: // on 64bit systems
export LD_LIBRARY_PATH=/opt/intel/fc/9.1.051/lib: // on 32bit systems

3.2. Ifort parameter bug

The ifort compiler v. 9.1.051 has a bug with public character parameters in modules.
Example:

CHARACTER (LEN=40), PUBLIC, PARAMETER :: c_att_name(2)= &
   / 'title ', &
   / 'history '/

The parameter c_att_name can easily be accessed from an external Fortran method. But if a Mono C# application calls a Fortran method, that accesses c_att_name, it will crash without error message. The solution is a Fortran function, that exports the parameter as a return value. Now the values can be accessed from C#.

PUBLIC FUNCTION get_c_att_name ( idx ) &
   RESULT(res)
   CHARACTER (LEN=40) :: res
   res = c_att_name(idx)
END FUNCTION get_c_att_name

3.3. Fortran interface functions

The <fortranEngine>.so interface functions are the same as in Windows Fortran. The two example functions are part of the module gei_ui and will be accessed from C# in 4.2.1.. The first one returns the integer comp_id_len.

FUNCTION gei_component_id_len ( comp_id_len ) RESULT( ok )
   INTEGER :: comp_id_len ! [OUT] len of the character string component id
   LOGICAL :: ok
   ...
END FUNCTION gei_component_id_len

The second example returns the character string comp_id.

FUNCTION gei_component_id ( comp_idx, comp_id ) RESULT( ok )
   INTEGER :: comp_idx ! [IN] component index
   CHARACTER (LEN=*) :: comp_id ! [OUT] component id
   LOGICAL :: ok
   ...
END FUNCTION gei_component_id

4. How to port a C# wrapper from Windows to Linux

4.1. General

The Mono Guidelines Interop with Native Libraries gives general information about the interface between managed and unmanaged code.

The wrapper has three layers:

- <engine>DllAccess.cs is the inner class for accessing the Fortran dll;
- <engine>DotNetAccess.cs references <engine>DllAccess.cs;
- <engine>Wrapper.cs is a LinkableComponent and the outer layer.

Compilation of the two inner layers:

```
```

Compilation of the outer layer:

```
```

### 4.2. How to port the individual layers

Fortunately, the original Windows C# code can nearly remain as it is.

#### 4.2.1. <engine>DllAccess.cs

<engine>DllAccess offers access to the Fortran shared library, e.g. @"gei.xe.so". The most of the Windows C# code remains unchanged and is displayed in black colour in the following example. The Fortran module gei_ui, method gei_component_id_len returns the integer comp_id_len.

```csharp
[DllImport(@"gei.xe.so"),
             EntryPoint = "gei_ui_mp_gei_component_id_len",
             SetLastError=true,
             ExactSpelling=true,
             CallingConvention=CallingConvention.Cdecl]
public static extern bool gei_component_id_len(ref int comp_id_len);
```

Some more adjustments are necessary, if a Fortran method returns the character string quantld:

```csharp
[DllImport(@"gei.xe.so"),
             EntryPoint = "gei_ui_mp_gei_out_exch_quant_id",
             SetLastError=true,
             ExactSpelling=true,
             CallingConvention=CallingConvention.Cdecl]
public static extern bool gei_out_exch_quant_id(ref int compIdx, ref int outExchangeItemN,
                                             [Out] byte[] quantld, uint lengthId);
```

#### 4.2.2. <engine>DotNetAccess.cs

Methods accessing int variables, e.g. ComponentIdLen(), remain unchanged from Windows:

```csharp
MarshalAs(UnmanagedType.LPStr) StringBuilder quantId
```

But on Linux systems StringBuilder can very rarely lead to variables with undefined return values. It is not clear why this happens. The Mono guidelines display a slightly different case, where the Mono garbage collector frees memory before the character string is returned, s. paragraph “GC-Safe P/Invoke code”.

However, the variables of type [Out] byte[] were always returned correctly and it is recommended to use them on Linux. The StringBuilder remains the better solution for Windows .NET. Developers are invited to find a solution that works on Windows as well as on Linux, e.g. an additional C / C++ wrapper or a SWIG generated wrapper.

#### 4.2.2. <engine>DotNetAccess.cs

Methods accessing int variables, e.g. ComponentIdLen(), remain unchanged from Windows:
public int ComponentIdLen()
{
    int compIdLen = 0;
    if((!(GEIDllAccess.gei_component_id_len(ref compIdLen))))
    {
        CreateAndThrowException();
    }
    return compIdLen;
}

This example displays that the variable of type byte[ ] from 4.2.1. is externally encoded to the string str:

public string OutputExchangeQuantityId(int compIdx, int outputExchangeN)
{
    byte[ ] quantId = new byte[QuantityIdLen()];
    // increment counter because array indices start in C# with 0
    // whereas in Fortran with 1
    int n1 = outputExchangeN + 1;
    if(!(!(GEIDllAccess.gei_out_exch_quant_id(ref compIdx, ref n1, quantId, (uint)QuantityIdLen()))))
    {
        CreateAndThrowException();
    }
    string str = Encoding.ASCII.GetString(quantId);
    return str.Trim();
}

Furthermore, the Initialize method of this class is a good place for a check, whether the dll is running on Mono or not:

if( Type.GetType("Mono.Runtime") == null )
{
    throw new Exception("this version of BAW.OpenMI.GEIDotNet.dll is only meant for use on Linux and Mono");
}

4.2.3. <engine>Wrapper.cs

There are no changes compared to Windows C#.

5. Use of the LinkableComponent

Before using the LinkableComponent its location must be known by Linux. It is recommended to put all shared libraries <fortranEngine>.so, <engine>DotNetAccess.dll and <engine>Wrapper.dll in one directory <componentDir>.

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<componentDir>

Now the generated LinkableComponent can be connected to any other LinkableComponent. Adding it as a model to the Linux version of the ConfigurationEditor is an easy test, s. How to port the OpenMI from Windows to Linux.