Enterprise Architect

User Guide Series

Hybrid Scripting

Author: Sparx Systems

Date: 6/10/2016

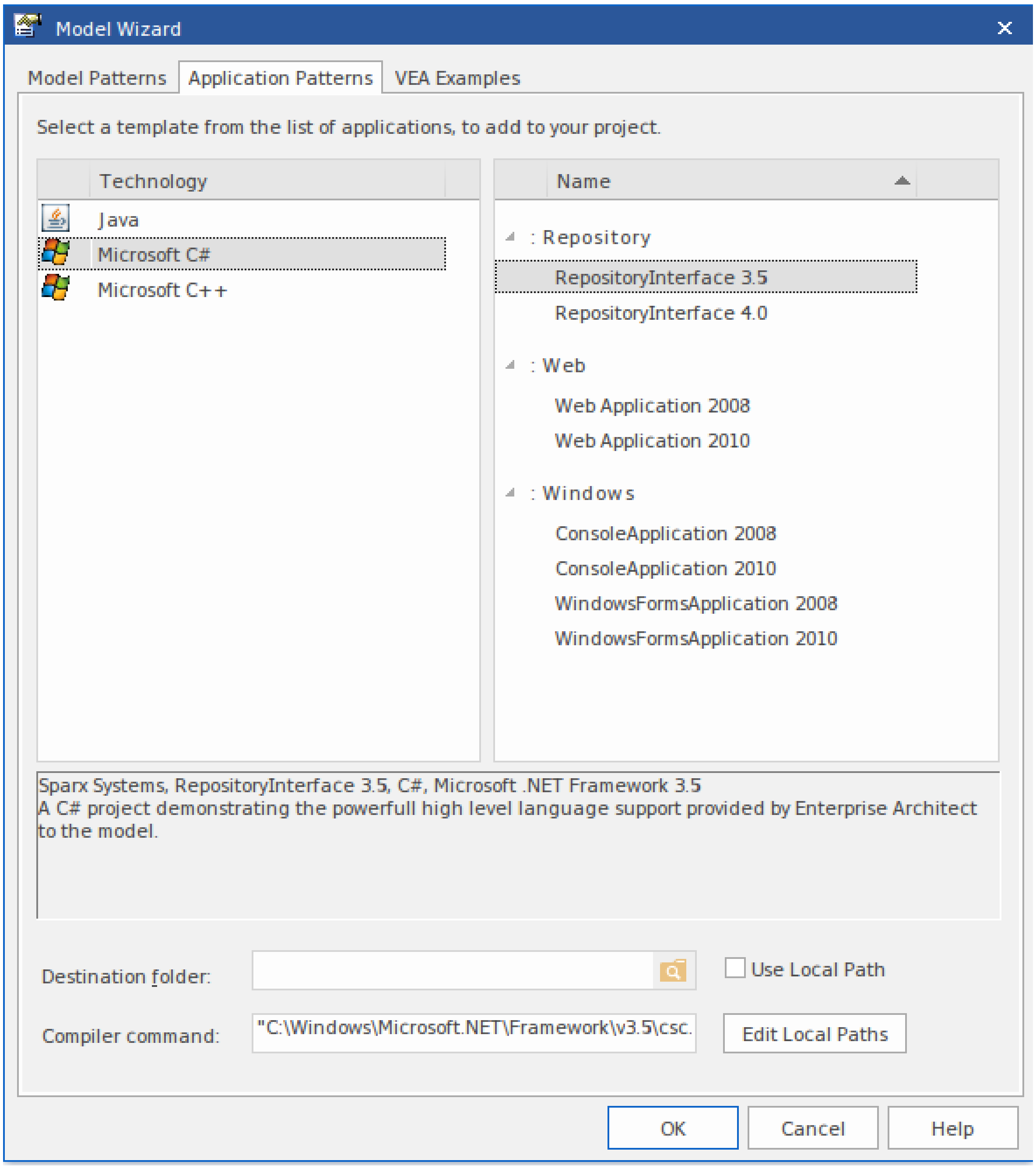
Version: 1.0

CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Hybrid Scripting

Hybrid scripting is provided to extend the capabilities of the standard scripting environment to high level languages such as Java and C#. Hybrid scripting provides a speed advantage over conventional scripting, and also allows authors to leverage existing skills in popular programming languages.



Select a template from the list of applications, to add to your project.

Sparx Systems, RepositoryInterface 3.5, C#, Microsoft .NET Framework 3.5
A C# project demonstrating the powerfull high level language support provided by Enterprise Architect to the model.

Destination folder:

Compiler command: "C:\Windows\Microsoft.NET\Framework\v3.5\csc.

Figure 1: The **Model Wizard**

Access

Access	Method

Ribbon	Design > **Model Wizard** > Application Patterns
Context Menu	Right-click on Package | Add a Model using Wizard | Application Patterns
Keyboard Shortcuts	**Ctrl+Shift+M** | Application Patterns
Other	**Project Browser** caption bar menu | New Model from Pattern | Application Patterns

Features

- Superior execution speed
- Enhanced interoperability
- Full **Visual Execution Analyzer** support

C# Example

Create the project

In the **Project Browser**, select the Package in which to create the template. With this Package selected, use the ribbon or context menu to bring up the **Model Wizard**. In the Model Wizard, open the 'Application Patterns' page. From this page select the C# *RepositoryInterface* template. (Note: You can choose from either the 4.5 or 4.0 framework versions) Enter the destination folder on the file system where the project template will be created, and click on the **OK button**.

Open the project

A Package structure similar to this should be created for you.

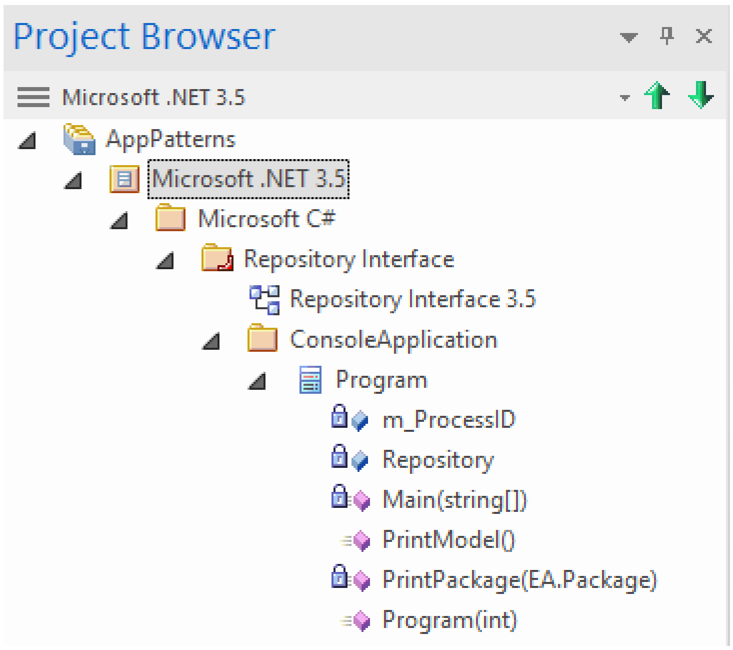
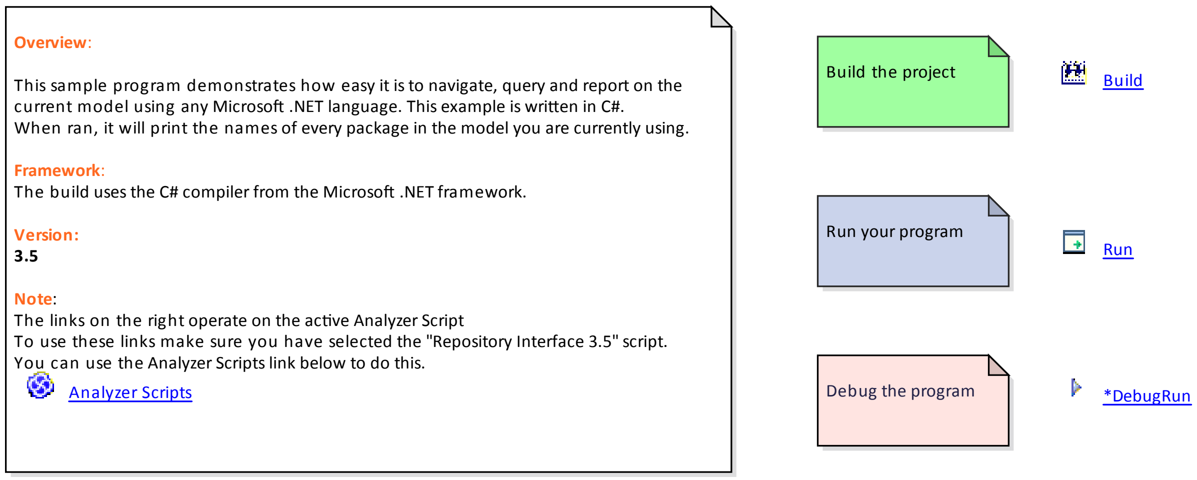


Figure 2: The C# Repository Project Structure

Expand the structure until you locate the Console Application diagram and open it.



Overview:

This sample program demonstrates how easy it is to navigate, query and report on the current model using any Microsoft .NET language. This example is written in C#.
When ran, it will print the names of every package in the model you are currently using.

Framework:
The build uses the C# compiler from the Microsoft .NET framework.

Version:
3.5

Note:
The links on the right operate on the active Analyzer Script
To use these links make sure you have selected the "Repository Interface 3.5" script.
You can use the Analyzer Scripts link below to do this.
Analyzer Scripts

Figure 3: The C# Repository Project Diagram

Build the script

The commands on this diagram will operate on the active build configuration. Before executing them, double-click on the *AnalyzerScripts* link and ensure the Repository Interface build configuration has a checkbox next to it.

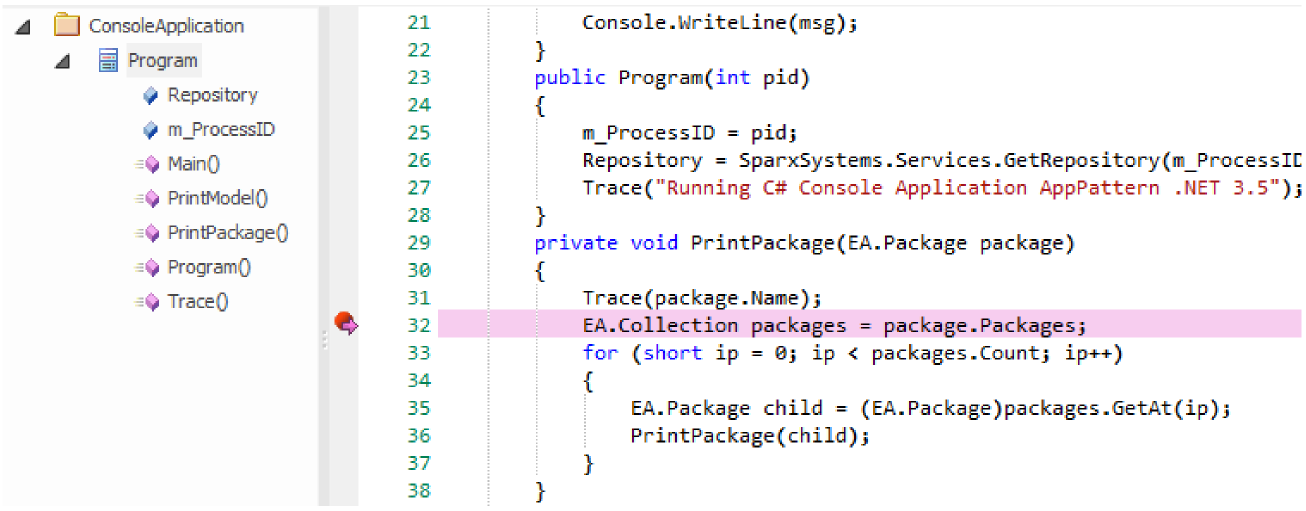
Run the script

Double click the 'Run' link. A Console should open. The Console will pause after completion so you can read the output. The output from the program will also be output to the 'Script' tab of the **System Output** window. You can alter this by changing the code.

Debug the script

Select the Program Class from the **Project Browser** and press **Ctrl+E** to open the source code.

Place a breakpoint in one of the functions and then double click the 'Debug' link. When the breakpoint is encountered, the line of code will become highlighted in the editor, as shown:



```
    Console.WriteLine(msg);
}
public Program(int pid)
{
    m_ProcessID = pid;
    Repository = SparxSystems.Services.GetRepository(m_ProcessI
    Trace("Running C# Console Application AppPattern .NET 3.5");
}
private void PrintPackage(EA.Package package)
{
    Trace(package.Name);
    EA.Collection packages = package.Packages;
    for (short ip = 0; ip < packages.Count; ip++)
    {
        EA.Package child = (EA.Package)packages.GetAt(ip);
        PrintPackage(child);
    }
}
```

Figure 4: Debugging the script

Java Example

Create the project

Select the package in the project browser where you wish the template to be created. With this package selected, use the ribbon or context menu to bring up the **Model Wizard**. From the Model Wizard, open the Application Patterns page. From this page select the Java *RepositoryInterface* template. Enter the destination folder on the file system where the project template will be created, and click OK.

Open the project

A package structure similar to that below should be created for you.

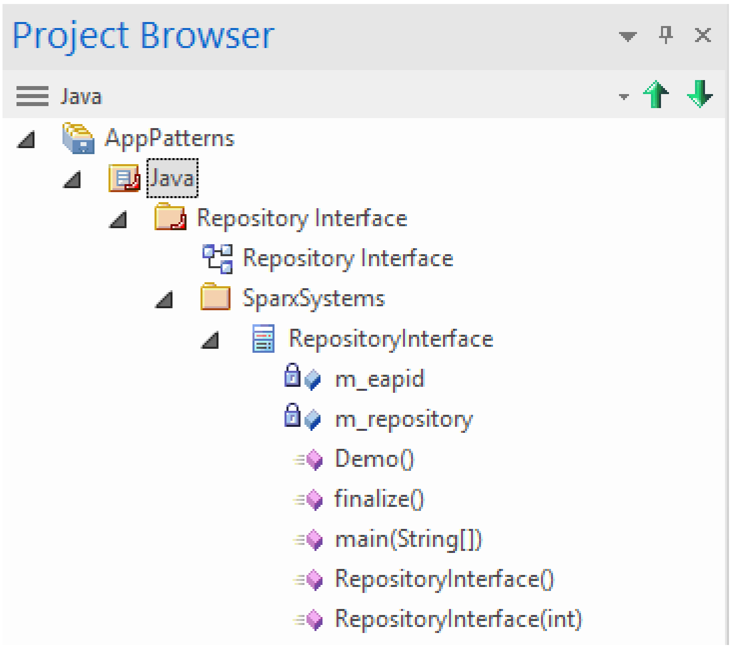
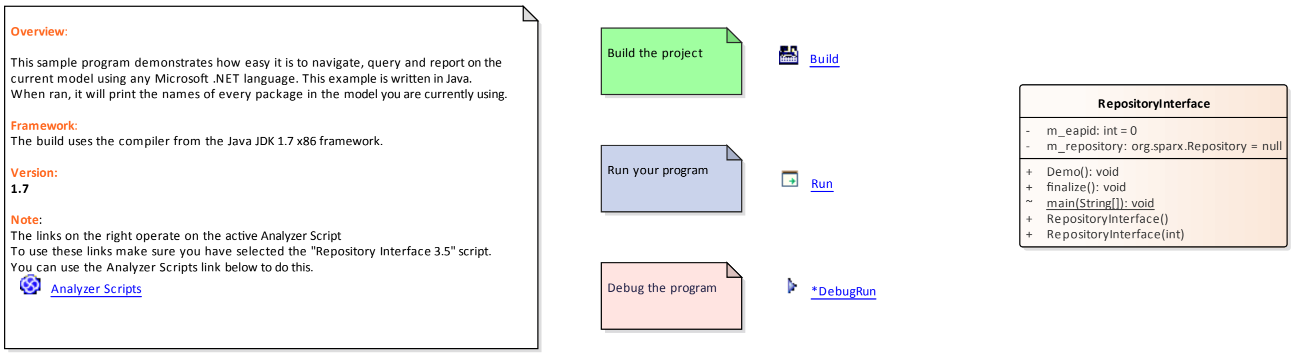


Figure 5: The Java Repository Project Structure

Expand the structure until you locate the RepositoryInterface diagram and open it.



Overview:

This sample program demonstrates how easy it is to navigate, query and report on the current model using any Microsoft .NET language. This example is written in Java.
When ran, it will print the names of every package in the model you are currently using.

Framework:
The build uses the compiler from the Java JDK 1.7 x86 framework.

Version:
1.7

Note:
The links on the right operate on the active Analyzer Script
To use these links make sure you have selected the "Repository Interface 3.5" script.
You can use the Analyzer Scripts link below to do this.
Analyzer Scripts

Build the project — Build
Run your program — Run
Debug the program — *DebugRun

RepositoryInterface

```
-   m_eapid: int = 0
-   m_repository: org.sparx.Repository = null

+   Demo(): void
+   finalize(): void
~   main(String[]): void
+   RepositoryInterface()
+   RepositoryInterface(int)
```

Figure 6: The Java Repository Project Diagram

Build the script

The commands on this diagram will operate on the active build configuration. Before executing them, double click the *AnalyzerScripts* link and ensure the Repository Interface build configuration has a check box next to it.

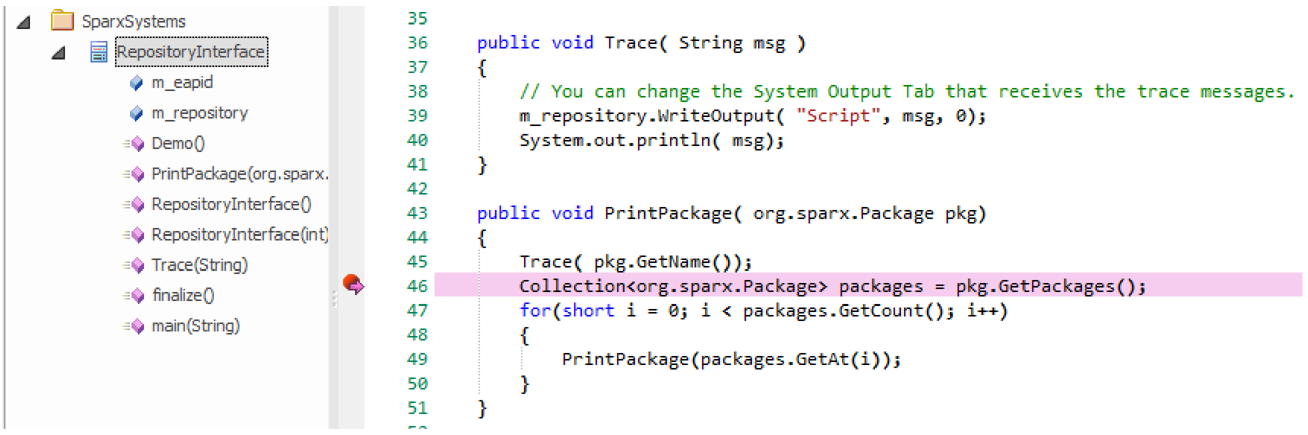
Run the script

Double click the Run link. A Console should open. The Console will pause after completion so you can read the output. The output from the program will also be output to the "Script" page of the "**System Output**" window. You can alter this by changing the code.

Debug the script

Select the Program class from the Project browser and click CTRL+E to open the source code.

Place a breakpoint in one of the functions and then double click the Debug link. When the breakpoint is encountered, the line of code will become highlighted in the editor like so.



```

public void Trace( String msg )
{
    // You can change the System Output Tab that receives the trace messages.
    m_repository.WriteOutput( "Script", msg, 0);
    System.out.println( msg);
}

public void PrintPackage( org.sparx.Package pkg)
{
    Trace( pkg.GetName());
    Collection<org.sparx.Package> packages = pkg.GetPackages();
    for(short i = 0; i < packages.GetCount(); i++)
    {
        PrintPackage(packages.GetAt(i));
    }
}
```

Figure 7: Debugging the script