Tech Note 566
Ensuring Accurate and Consistent Deployment of Custom Script Function Libraries

Topic#: 002318
Created: January 2009

Introduction

Wonderware Application Server provides you the ability to leverage the .NET development language for custom and advanced functionality made available through the ArchestrA Integrated Development Environment (IDE).

Custom Script Function Libraries are integrated with Wonderware Application Server by

- Importing multiple, independent DLLs, or;
- Importing multiple DLLs that have dependencies into the Galaxy.

Wonderware Application Server accommodates importing of multiple DLLs and DLLs with dependencies through the use of **aaSLIB** files. **aaSLIB** files make managing multiple DLLs and their dependencies easier by eliminating the need to track multiple DLLs.

Working Definitions

This *Tech Note* uses terms that may be familiar but which can be used in different ways. The following definitions are applied in the context of this document:

- A **Custom DLL** (Shared Library) is simply compiled code that can be accessed from scripting or from a .NET control. A DLL is a module that contains functions and data that can be used by another module (application or DLL).

  It can also be described as a collection of small programs, any of which can be called when needed by a larger program that is running in the computer. The small program that lets the larger program communicate with a specific device such as a printer or scanner is often packaged as a DLL program (usually referred to as a DLL file). DLL files that support specific device operation are known as device drivers.

Source Citation

In this context, the custom DLLs provide added functionality for use by Wonderware Application Server.

Note: The Custom DLL is not to be confused with a .NET Client Control. The import and management of .NET Client Controls is different, and will be addressed in a future *Tech Note*.

Application Versions

- Wonderware Application Server 3.0 and later.

Quick Links

Summary Workflow

The following steps outline best practice for ensuring all script function libraries are correctly managed in the Galaxy.

1. Document all the DLLs used in the Galaxy in a table format.
2. For each DLL, identify if it is either a top-level DLL or a dependent DLL or both in different scenarios as the case may be.

> **Note:** If the dependencies between DLLs is not explicitly known, the **.NET Reflector** tool can be used to discover them. This free tool can be downloaded from **Red Gate software**.

3. For each top-level DLL, identify the dependent DLLs that are needed for the top-level DLL to function correctly. These top-level DLLs with dependencies will determine the number of aaSLIB files that will need to be created.
4. Create **aaSLIB** files for each top-level DLL. Make sure you correctly reference the dependent DLLs as noted the **aaSLIB creation process**.

   Only the top-level DLL defined in the aaSLIB package will have all of its exposed methods available from the Script Function Browser during script development.

> **Note:** For details about creating and referencing dependent DLLs, see **Tech Note 518 - Managing Custom Script Function Libraries for Use in Application Server** or alternatively refer to the section in this document called Working with aaSLIB Files.

5. Import one-by-one each stand-alone DLL and each aaSLIB file into the Galaxy using the **Import/Script Function Library** main menu commands in the IDE.

6. Once all the needed stand-alone DLLs or aaSLIB files have been imported, ensure to validate all objects that contain script references to the imported script function libraries. This step allows verification that script references to the imported script function libraries resolve correctly.

Note: If you are having trouble getting an object to validate correctly immediately after importing a script function library, try rebooting the GR Node.

7. Deploy the objects that contain the script references to the imported script function libraries.

8. Test & debug the scripts to ensure expected behavior.

Managing Custom Script Function Library Imports

You can import a single DLL, or a collection of DLLs packaged together. You use the **Import/Script Function Library** main menu command of the IDE In both cases to import either the single, or the top-level DLL.

When you import a script function library/DLL, Wonderware Application Server creates two files, and writes to the GR Database, regardless of whether you import a single DLL, or multiple DLLs packaged together via an aaSLIB file.

When you import a single DLL called **<Name>.dll**, Application Server does the following:

- Application Server creates a set of reference files in the following location on the GR Node:

  <drive>:\Program Files\ArchestrA\Framework\FileRepository\<GalaxyName>\AddOns\QScript2Lib

- Application Server creates an aaSLIB file with the same name, such as <Name>.aaSLIB
- Application Server creates an XML file with the same name, such as <Name>.xml
- Application Server copies the DLL to the following directory on the GR. This makes the DLL available for copy to deployed platforms in the Galaxy. These are the Platforms that host objects with script references to the DLL.

  <drive>:\Program Files\ArchestrA\Framework\FileRepository\<GalaxyName>\Vendors\ArchestrA

Note: <drive> is the drive letter that was selected to install Wonderware Application Server during its initial installation.

Importing a Single DLL

- Use the **Import/Script Function Library** main menu command of the IDE and import the single stand-alone DLL.

After the import, this DLL and all of its exposed methods are available from the Script Function Browser when you create scripts in the IDE Script Editor.

Importing Multiple DLLs & Working with Their Dependencies

Before importing a DLL set, you must create an **aaSLIB** file that captures all of that DLL's dependent DLLs for *every* custom DLL referenced in a script. It is highly recommended to use a blank galaxy to import and export the top level DLLs. This is because Application Server does not natively support removing previously-imported script function libraries.

1. To create this file, you must import a top-level DLL using the **Import/Script Function Library** main menu command of the IDE.
2. Export the top-level DLL as an **aaSLIB** file using **Export/Script Function Library**. The aaSLIB file groups a set of DLLs together and defines the dependent relationships between them into one single package. The relationships between the top-level DLLs and their dependencies are defined using an XML file contained in the aaSLIB package.

Application Server does the following when you import an aaSLIB package called **<Name>.aaSLIB**:

- Creates a set of reference files in the following location:

  <drive>:\Program Files\ArchestrA\Framework\FileRepository\<GalaxyName>\AddOns\QScript2Lib

- Creates an aaSLIB file of the same name, such as **<ScriptLibraryName>.aaSLIB**
- Creates an XML file of the same name, such as **<ScriptLibraryName>.xml**
- Copies the extracted contents of the aaSLIB file (containing all the DLLs) to the following directory on the GR. The collection is available for copy to deployed Platforms in the Galaxy (platforms that host objects with script references to the top level dll defined in aaSLIB file).

  <drive>:\Program Files\ArchestrA\Framework\FileRepository\<GalaxyName>\Vendors\ArchestrA

Refer to this location as the **MASTER** Copy, from which all other deployed platforms will receive a copy where required.

Note: In addition to the copied files mentioned above, Wonderware Application Server also writes entries to the Galaxy Database in order to record the imported stand-alone DLLs or aaSLIB files. Refer to the section **Removing Script Function Libraries** for more information on this aspect.

Even though the top level DLLs and their dependencies are defined, a file copy of these DLLs to a platform (remote or locally on the GR) *only* occurs if there is a scripting reference to the top level DLL in a deployed object (for example the object can be a Platform, Engine, Area or User Defined Object).

In other words, the files only get copied when a deployed object references a function call in one or more of the top level DLLs.

Example

An aaSLIB file must be created for every single Top Level (referring) DLL.

Let's say we have three DLLs called **D1**, **D2**, and **D3**, and their relationship is as follows:

- D1 depends on D2 and D3
- D2 depends on D3

How many aaSLIB files must be created and imported using the IDE?

Two, one for each referring DLL:

- **D1.aaSLIB**, which references D2, D3
- **D2.aaSLIB**, which references D3

The working assumption is that you are also using methods from D3 in Application Server Scripts. In this case, what do you import using the IDE?

- D1.aaSLIB
- D2.aaSLIB
- D3.dll

Working with aaSLIB Files

Use aaSLIB files to manage a group of DLLs with dependencies.

Creating the aaSLIB File

Create the aaSLIB files using the following steps.

1. To create this file, you must import the top-level DLL using the **Import/Script Function Library** main menu command of the IDE.
2. Export the top-level DLL as an **aaSLIB** file using **Export/Script Function Library**. The aaSLIB file groups a set of DLLs together and defines the dependent relationships between them into one single package. The relationships between the top-level DLLs and their dependencies are defined using an XML file contained in the aaSLIB package.

   You can modify the file using Notepad or a similar XML editor to list the dependent DLLs following the steps in **Tech Note 518**.

You can also use the **aaSLIB Editor Utility** to define the dependent relationships between the script function libraries and group them together into a single package (aaSLIB file). Using the Editor Utility eliminates possible spelling and XML form errors. You can also designate specific DLLs for registration. The tool is Application Server-independent and automates many of the manual steps documented in the Tech Note.

Download the utility from **ArchestrA.biz** and install it. The utility requires the MS .NET Version 1.1 of the .NET Framework. If the correct .NET version is not on your computer, the utility directs you to the Microsoft Download location.

Note: The aaSLIB utility is a custom utility that is used *only* in this context to assist packaging related DLLs. **It is not officially supported by Wonderware Technical Support.**

Using the aaSLIB Utility

You must have a aaSLIB file to begin. This is provided when you import/export the DLL as aaSLIB.

1. Start the aaSLIB Editor using the desktop icon.
2. Use the **Browse** button to locate the aaSLIB file. When you locate the target file, the aaSLIB Editor creates a temporary directory on the C:\ drive called **wwTS** and copies the files to that location.

   Use the check the boxes only to register the DLL as COM if necessary. The checkboxes do not affect dependencies and are used only to register the DLL(s).

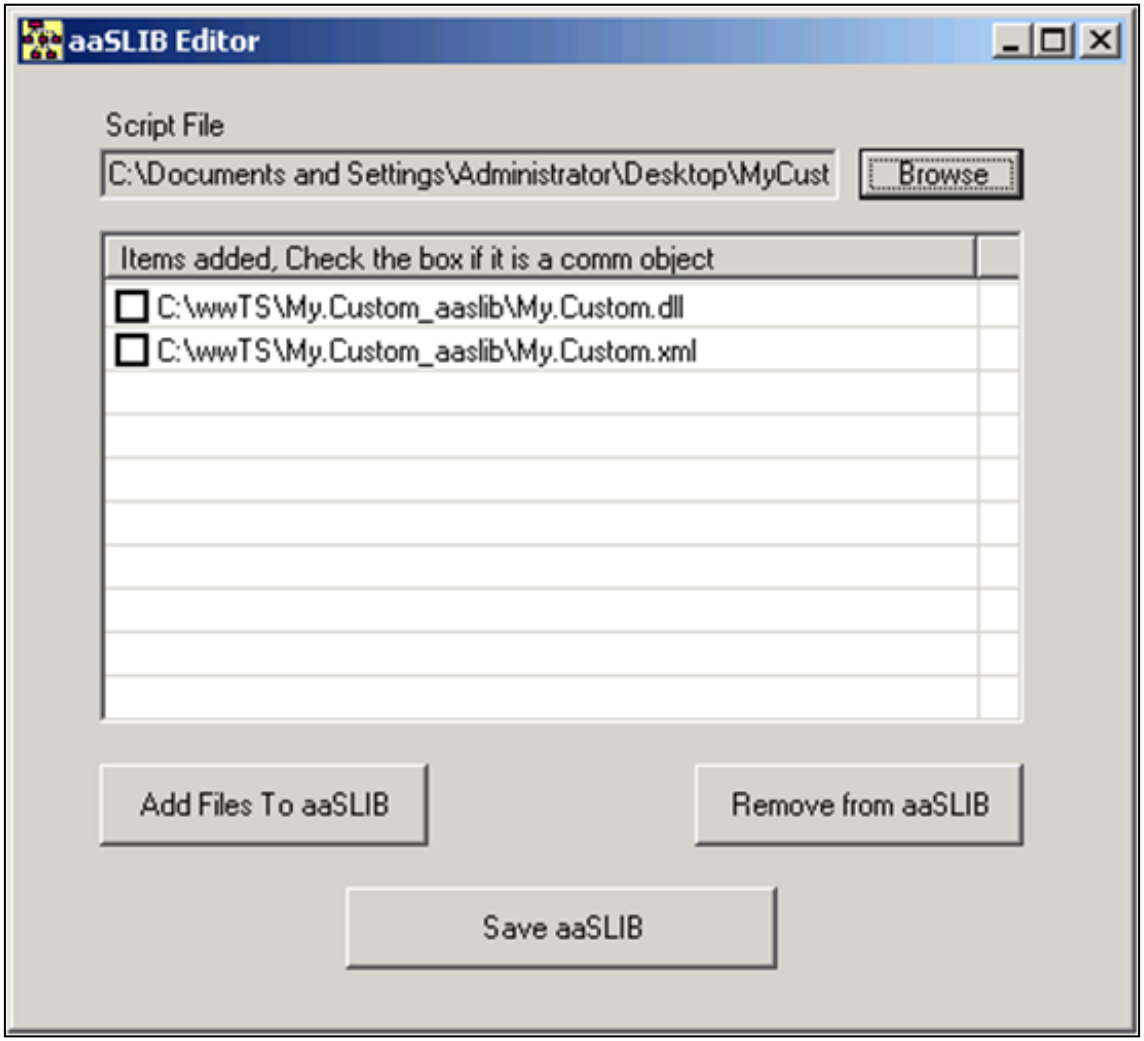

Figure 1: aaSLIB Editor Interface

3. When you locate the DLL, any DLLs in the library are displayed in the Editor. The first time you do this, only the Main DLL will be displayed.

4. Add additional DLLs that are dependencies of the main DLL. To add dependent DLLs, click the **Add Files To aaSLIB** button and locate the DLLs you need to include as dependencies. They appear below the original, top-level items as shown in the following graphic.

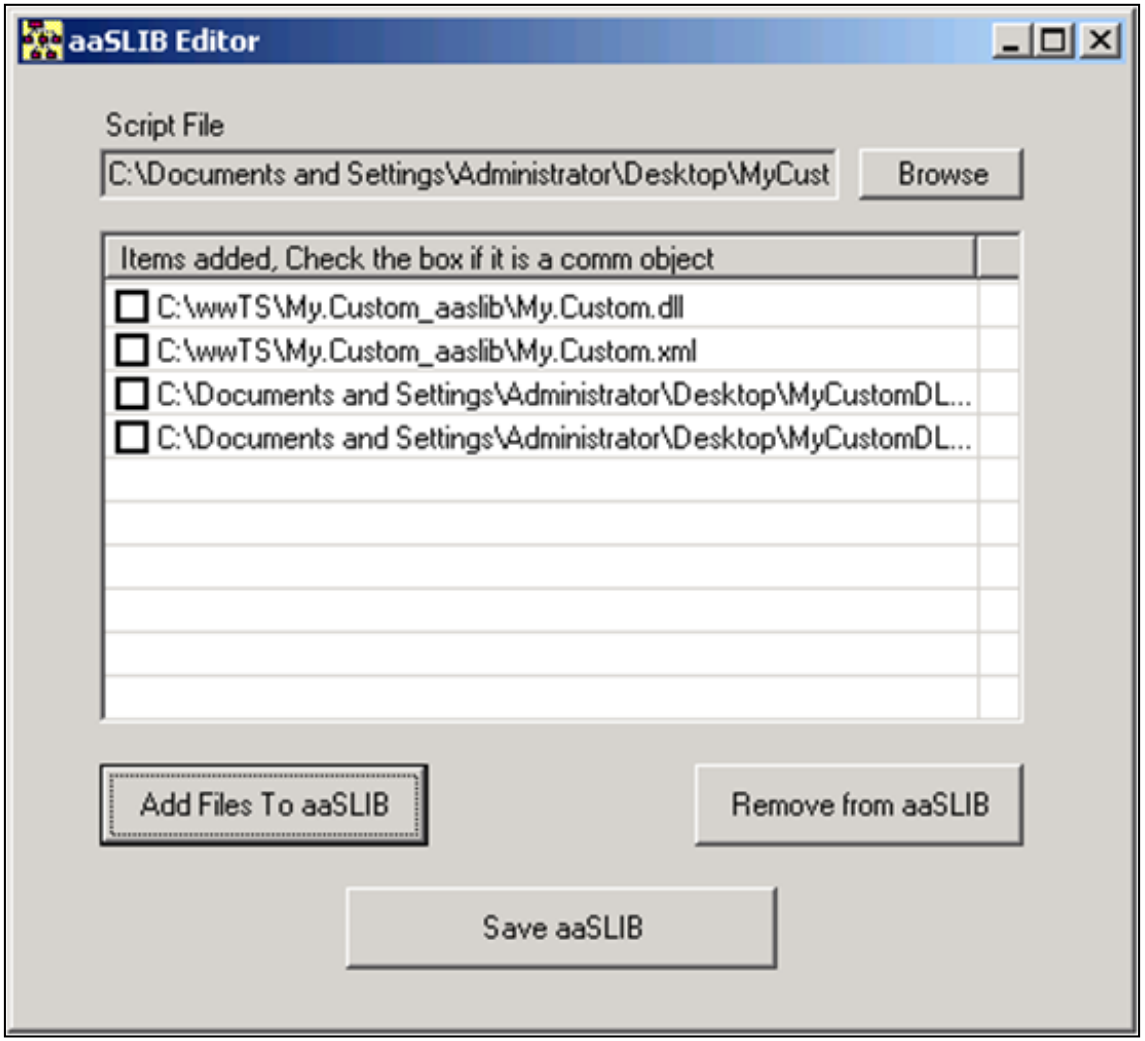

Figure 2: Dependent DLLs for My.Custom.dll

If any of the DLLs are COM, make sure that you click the check box beside the DLL – this will cause the COM DLL to be registered correctly.

You can also remove DLLs as required by highlighting the item and clicking **Remove from aaSLIB**. Note that the first items are always the top-level DLL, and cannot be removed.

5. Click **Save aaSLIB**. The editor renames the old (original) script function library and stores the new aaSLIB file in the same folder.

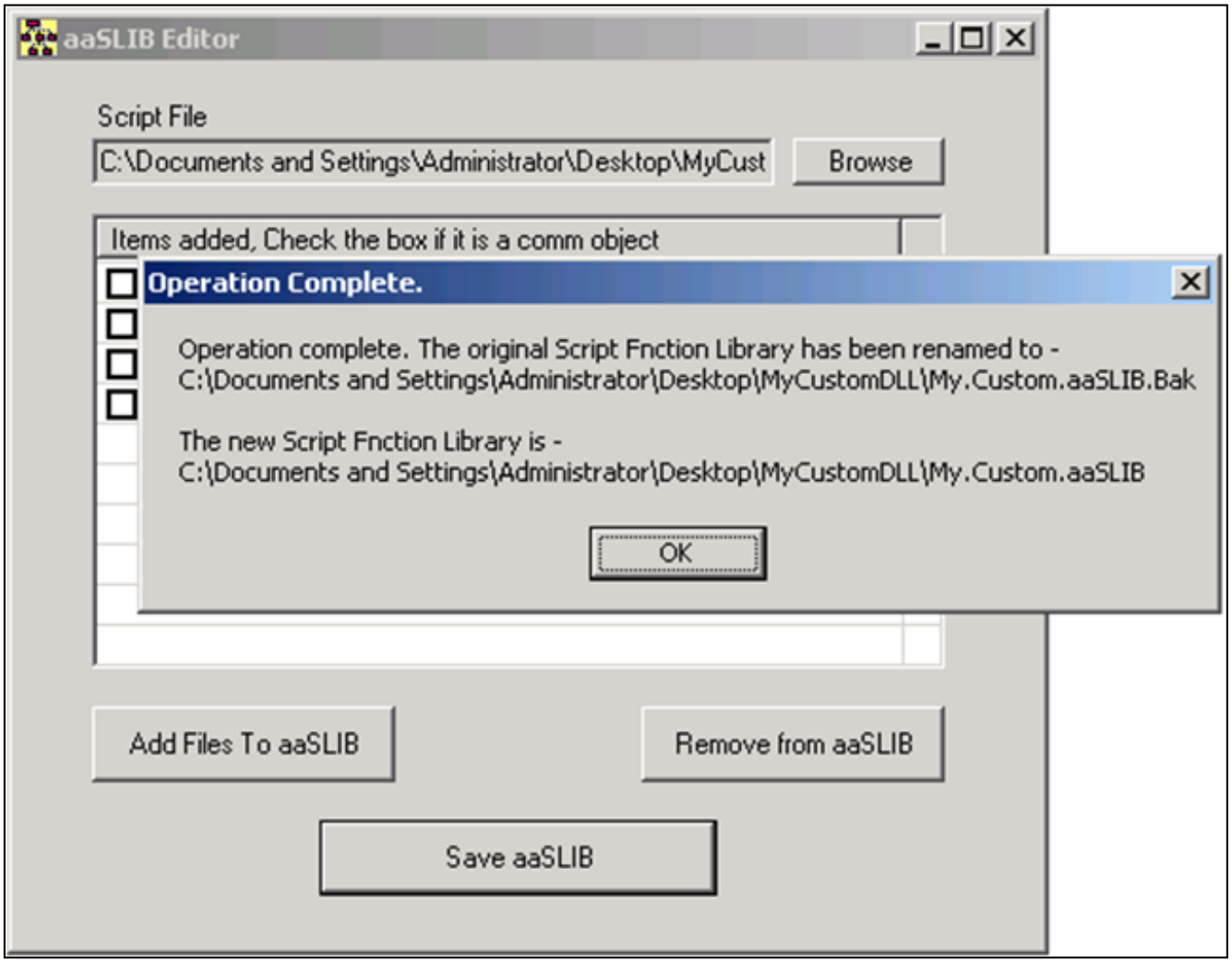

Figure 3: Operation Complete

6. Click **OK**, then close the Editor.

You can now import the newly created aaSLIB file into a new test Galaxy to test with your Application Server scripts.

Working with the Imported Libraries in Application Server

This section describes where you can access Script Functions in Application Server, and to show the default available script functions before you import any custom script function libraries via stand-alone DLLs or aaSLIB files.

Figure 4 (below) shows the list of available Microsoft Script Methods in the Scripting Editor's Script Function Browser.

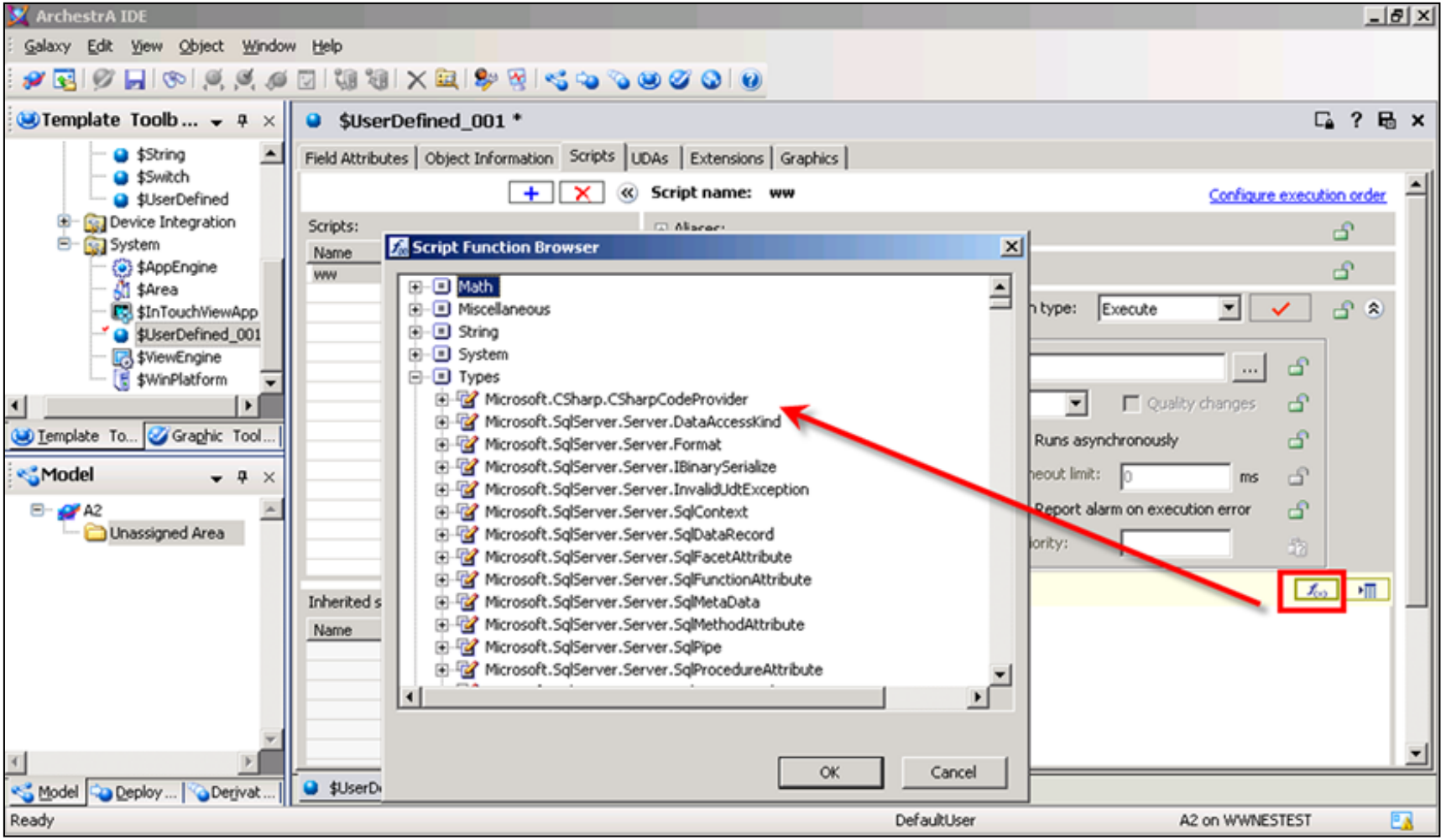

Figure 4: Default Script Methods Available in the IDE

To use your custom Script Functions, import each stand-alone DLL or aaSLIB file, one-by-one, into the Galaxy via the IDE.

In this example, importing the **.aaFactMES.aaSLIB** file displays the newly-available methods in the Script Function Browser. Any methods or functions from custom DLLs that are imported into the galaxy are displayed alphabetically in the Script Function Browser under the **Types** root.

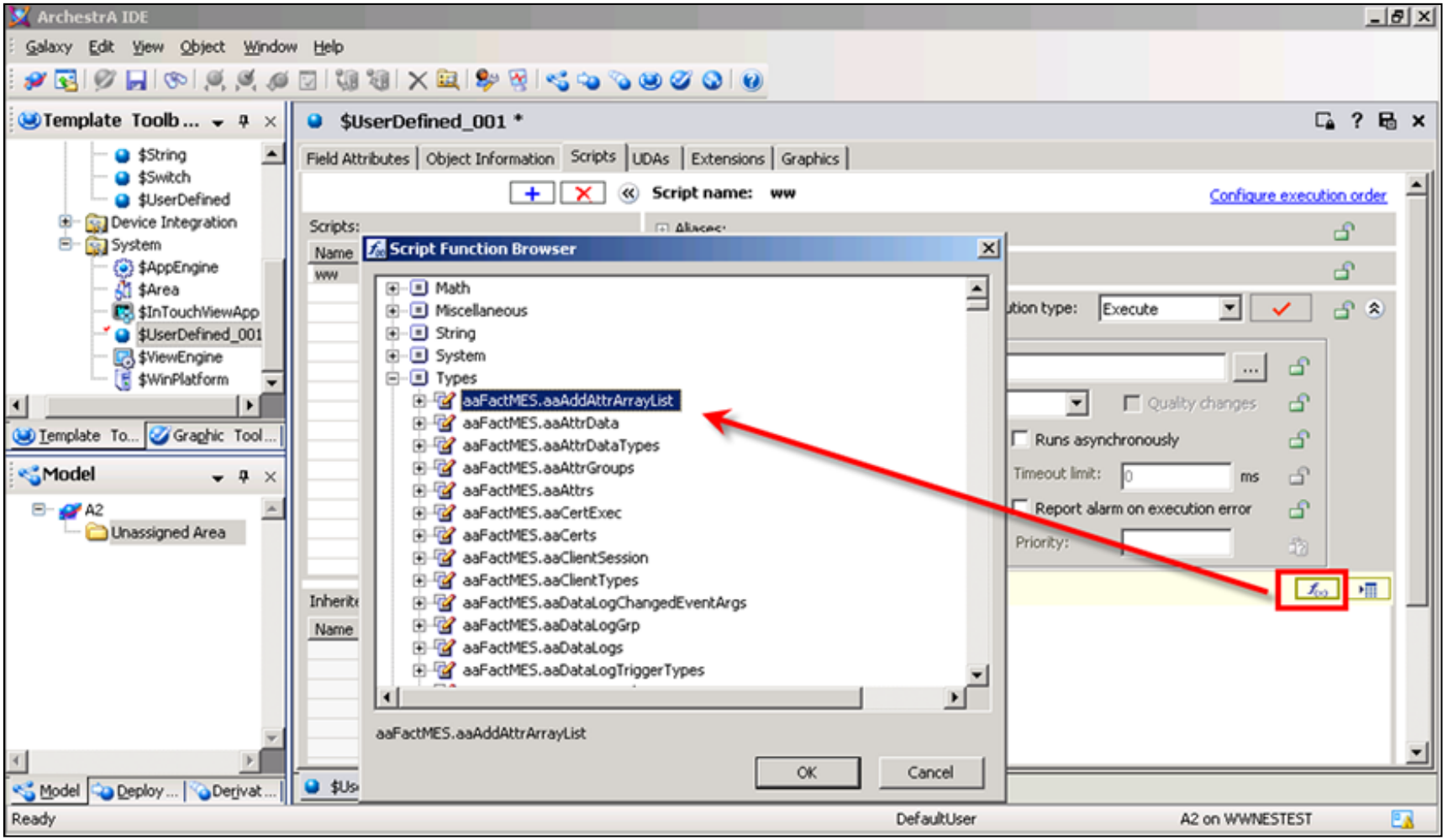

Figure 5: Custom Scripts Available from Types Root

Summary Script Function Library Import Recommendations

- When you need to import a single or multiple DLLs that do not have dependencies, use the **Import/Script Function Library** from the IDE.
- When you need to import a single or multiple DLLs that do have dependencies, you must first compile them into an aaSLIB package by importing, then exporting the top-level DLL.

Note: When you import an aaSLIB file, Application Server unpackages the file and places all of its contents into this folder on the GR Node:

<Drive>:\Program Files\ArchestrA\Framework\FileRepository\<GalaxyName>\Vendors\ArchestrA

Removing Script Function Libraries

It can be necessary to remove custom DLLs from a galaxy. In one scenario, an aaSLIB script library is imported into the Demo galaxy from the SQLData Components package. However, it is decided the project does not require the library, and the customer now wants it removed from the Galaxy.

In another scenario, it can be necessary to restore a system to a clean known state (no imported DLLs) before making controlled and monitored changes without impacting the rest of the Galaxy structure.

In order to accomplish these tasks, the files and database entries on the Galaxy Repository node must be deleted in addition to unloading any existing DLLs from memory.

For detailed steps to remove the Script Function Libraries, refer to **Tech Tip 0810: Removing Custom .NET Libraries from Industrial Application Server 3.0**. The Tech Tip contains information for removing the files, both manually and using a Stored Procedure.

References

- ***Tech Note* 568 How Wonderware Application Server Manages DLLs and aaSLIB Imports**
- ***Tech Note* 569 Using Merge Modules**
- **FAQ 38 Importing Custom DLLs**

S. Kermani

Tech Notes are published occasionally by Wonderware Technical Support. Publisher: Invensys Systems, Inc., 26561 Rancho Parkway South, Lake Forest, CA 92630. There is also technical information on our software products at **Wonderware Technical Support.**

For technical support questions, send an e-mail to **support@wonderware.com**.

Back to top