Tech Note 568

How Wonderware Application Server Manages DLLs and aaSLIB Imports

Topic#: 002320
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.

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 that were noted during the aaSLIB creation process described in **Tech Note 566 Ensuring Accurate and Consistent Deployment of Custom Script Function Libraries**.

   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.

How Application Server Manages DLLs and aaSLIB Imports

When you import a script function library into a Galaxy, Wonderware Application Server completes the following operations to track it at various stages of working with the imported DLLs.

- **Database Writes**: Writes to the Galaxy Repository (GR) database.
- **File Copy**: Creates reference files on the GR Node. When deploying objects with scripts with references to imported script function libraries to remote platform servers, Application Server copies the DLLs to the remote platform from the

GR Node.

- **Load into Memory**: When setting objects with scripts with references to imported script function libraries on-scan on remote Platform servers, Application Server will load the required DLLs into memory.

Let's take a look at each operation in more detail.

Database Writes

Reference **Tech Tip 0810: Removing Custom .NET Libraries from Industrial Application server 3.0**. This reference courtesy of Wonderware Northeast.

File Copy

When you import a script function library, Wonderware Application Server creates two files, regardless of whether you import a single DLL, or multiple DLLs packaged together via a aaSLIB file.

When you import a single DLL called <Name>.dll:

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

  <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 (with script references to the DLL):

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

When you import an aaSLIB package called <Name>.aaSLIB:

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

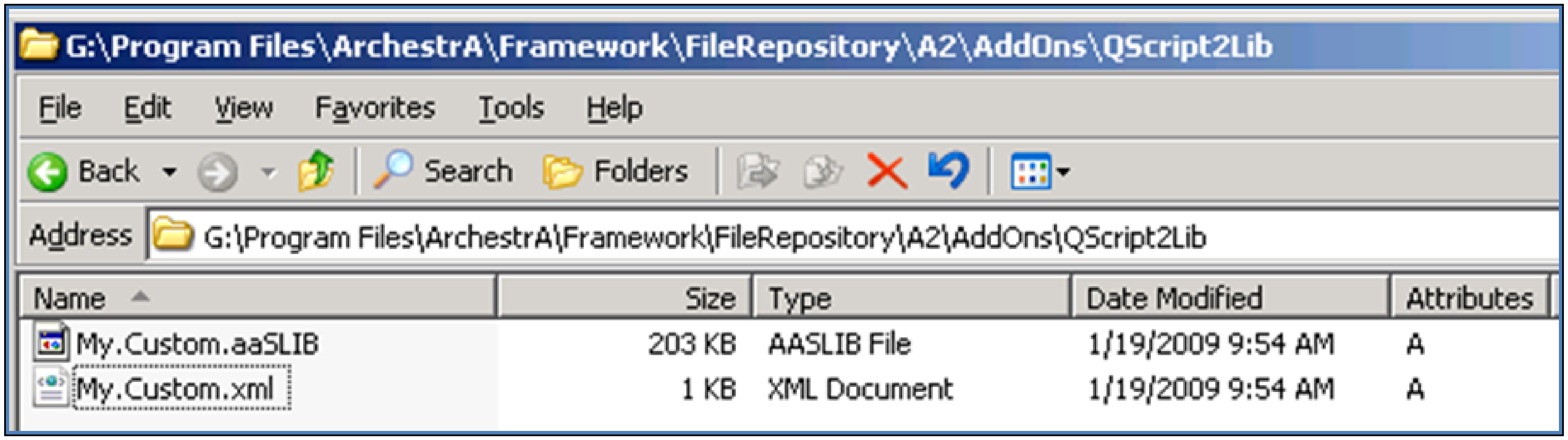
  <drive>:\Program Files\ArchestrA\Framework\FileRepository\<GalaxyName>\AddOns\QScript2Lib

- Application Server creates an aaSLIB file of the same name, such as **<Name>.aaSLIB**

- Application Server creates an XML file of the same name, such as **<Name>.xml**

- Application Server copies the extracted contents of the aaSLIB file (containing all the DLLs) to the following directory on the GR to be made available for copy to deployed platforms in the galaxy (with script references to the top level dll in the aaslib file).

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


G:\Program Files\ArchestrA\Framework\FileRepository\A2\AddOns\QScript2Lib
File
Edit
View
Favorites
Tools
Help
Back
Search
Folders
Address

G:\Program Files\ArchestrA\Framework\FileRepository\A2\AddOns\QScript2Lib

Name	Size	Type	Date Modified	Attributes
My.Custom.aaSLIB	203 KB	AASLIB File	1/19/2009 9:54 AM	A
My.Custom.xml	1 KB	XML Document	1/19/2009 9:54 AM	A

Figure 1: Files Delivered to QScript2Lib

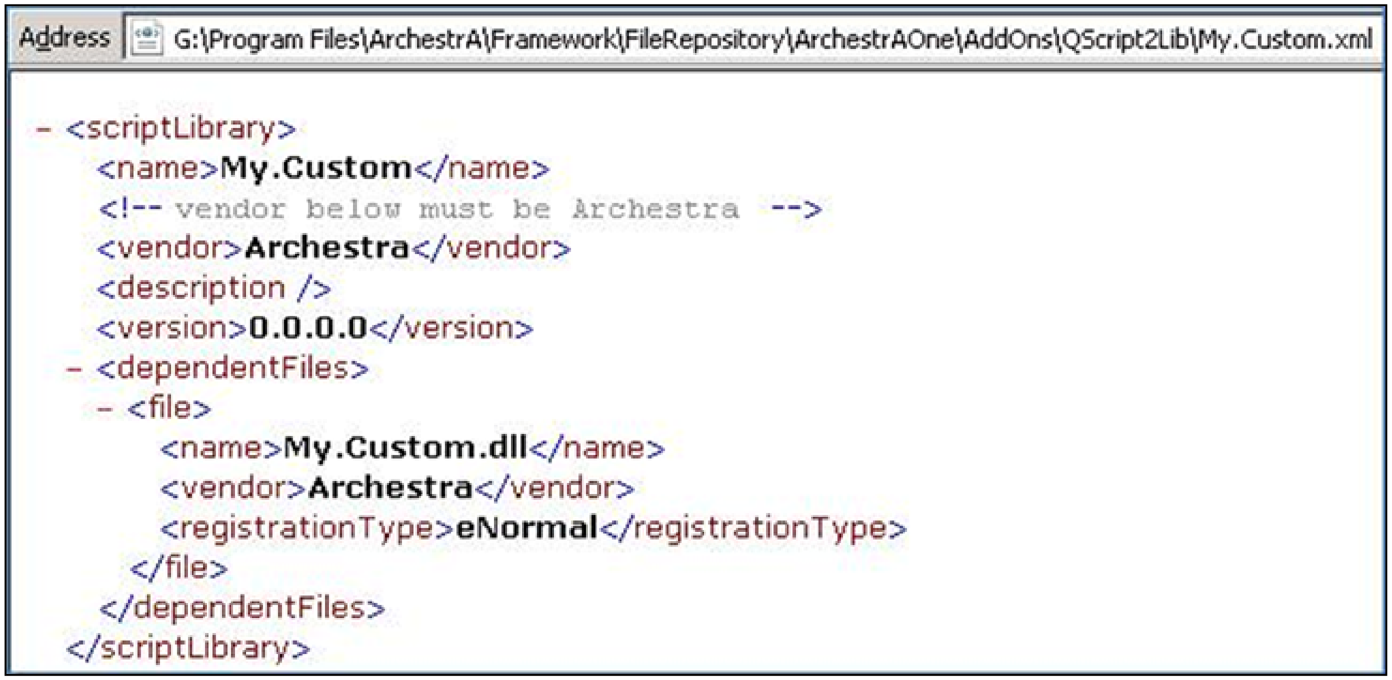
What is the difference between the two cases?

The key difference lies in the formation of the XML file that Application Server auto-generates when importing the stand-alone DLL or the aaSLIB package.

The XML file contains the structure that identifies any dependent DLLs. If dependent DLLs are listed, their relationship to the top level DLLs is defined there.

Here is an example XML file of a *single* imported DLL called **My.Custom.dll**.

In this example, this DLL has dependencies, but they are not shown, since we only import the single DLL. Wonderware Application Server sees it as the only top-level, and only DLL. This means that when you export this DLL, its dependencies will not be included, and the DLL will not function correctly on the target platform.

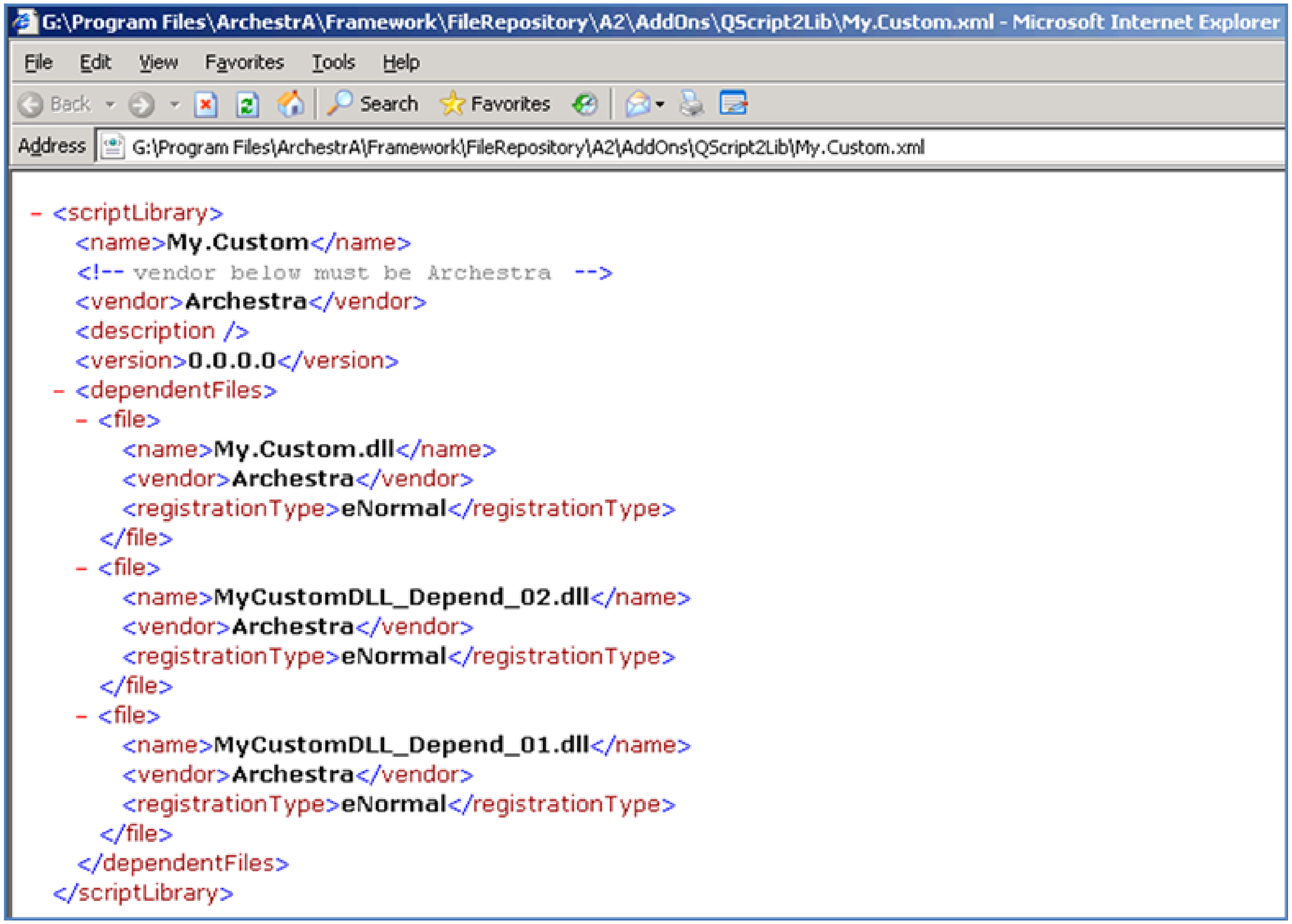


Address G:\Program Files\ArchestrA\Framework\FileRepository\ArchestrAOne\AddOns\QScript2Lib\My.Custom.xml

```
- <scriptLibrary>
    <name>My.Custom</name>
    <!-- vendor below must be Archestra  -->
    <vendor>Archestra</vendor>
    <description />
    <version>0.0.0.0</version>
  - <dependentFiles>
    - <file>
        <name>My.Custom.dll</name>
        <vendor>Archestra</vendor>
        <registrationType>eNormal</registrationType>
      </file>
    </dependentFiles>
  </scriptLibrary>
```

Figure 2: My.Custom.xml File Content

Here is an example XML file of an **aaSLIB** imported package of the same name, which has dependencies. The dependencies are shown here since we are importing an aaSLIB file, in which these dependencies have been identified.


G:\Program Files\ArchestrA\Framework\FileRepository\A2\AddOns\QScript2Lib\My.Custom.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address G:\Program Files\ArchestrA\Framework\FileRepository\A2\AddOns\QScript2Lib\My.Custom.xml

```
- <scriptLibrary>
    <name>My.Custom</name>
    <!-- vendor below must be Archestra  -->
    <vendor>Archestra</vendor>
    <description />
    <version>0.0.0.0</version>
  - <dependentFiles>
    - <file>
        <name>My.Custom.dll</name>
        <vendor>Archestra</vendor>
        <registrationType>eNormal</registrationType>
      </file>
    - <file>
        <name>MyCustomDLL_Depend_02.dll</name>
        <vendor>Archestra</vendor>
        <registrationType>eNormal</registrationType>
      </file>
    - <file>
        <name>MyCustomDLL_Depend_01.dll</name>
        <vendor>Archestra</vendor>
        <registrationType>eNormal</registrationType>
      </file>
    </dependentFiles>
  </scriptLibrary>
```

Figure 3: My.Custom.dll with Dependent DLLs

How the aaSLIB File Works

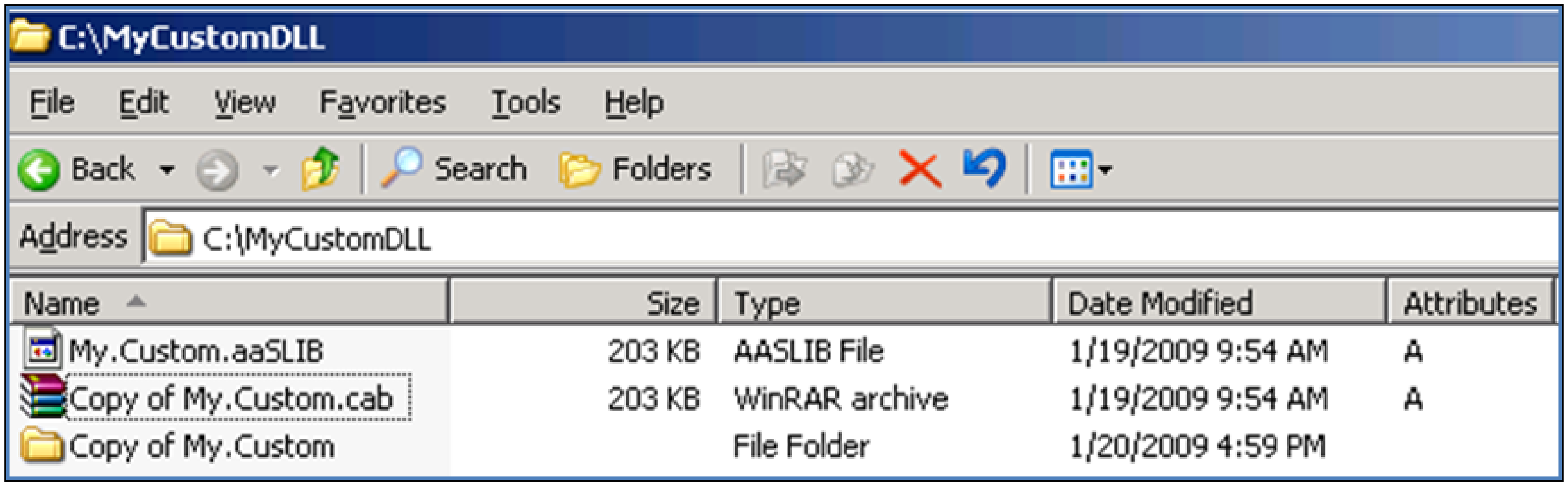
To better and further understand how the aaSLIB file works, we can open it. aaSLIB is a custom extension, but the file is a compressed Microsoft Cabinet (cab) file.

1. Make a copy of the aaSLIB file and rename the aaSLIB file extension to **.cab**.
2. Use any cabinet compression program to open and extract the contents (such as **WinRar**).

   Once extracted we should see the following generalized structure:

   - A **"__wwCabinetFileList.xml"** file.
   - One or more files with a ***.tmp** file extension.

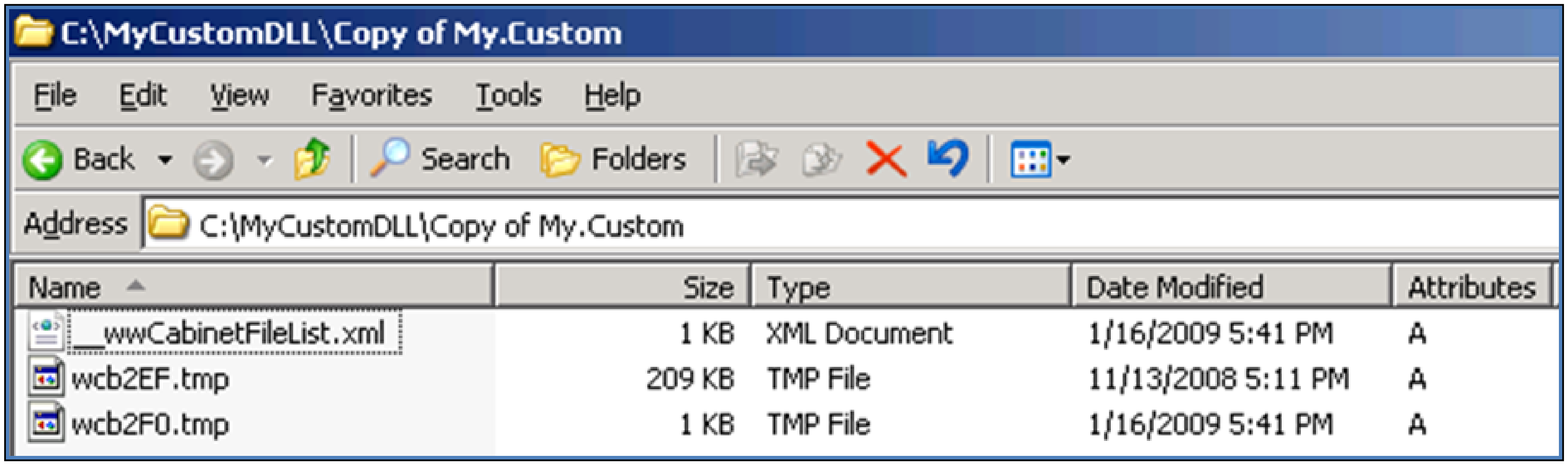
   Figure 4 (below) shows a copy of My.Custom.aaSLIB file renamed to .cab and then extracted using WinRar.



Name	Size	Type	Date Modified	Attributes
My.Custom.aaSLIB	203 KB	AASLIB File	1/19/2009 9:54 AM	A
Copy of My.Custom.cab	203 KB	WinRAR archive	1/19/2009 9:54 AM	A
Copy of My.Custom		File Folder	1/20/2009 4:59 PM	

Figure 4: aaSLIB File Contents

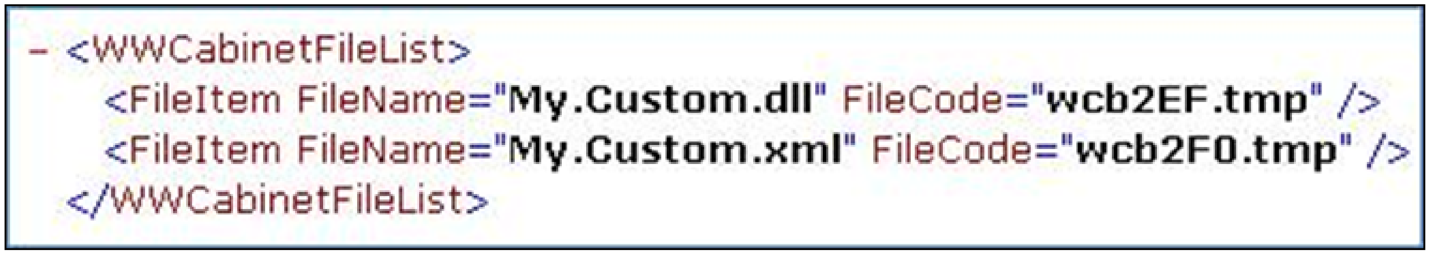
Figure 5 (below) shows the contents of the extracted cab file for the **My.Custom.aaSLIB** file (without any dependencies).


C:\MyCustomDLL\Copy of My.Custom
File Edit View Favorites Tools Help
Back Search Folders
Address C:\MyCustomDLL\Copy of My.Custom

Name	Size	Type	Date Modified	Attributes
__wwCabinetFileList.xml	1 KB	XML Document	1/16/2009 5:41 PM	A
wcb2EF.tmp	209 KB	TMP File	11/13/2008 5:11 PM	A
wcb2F0.tmp	1 KB	TMP File	1/16/2009 5:41 PM	A

Figure 5: Cab File Contents

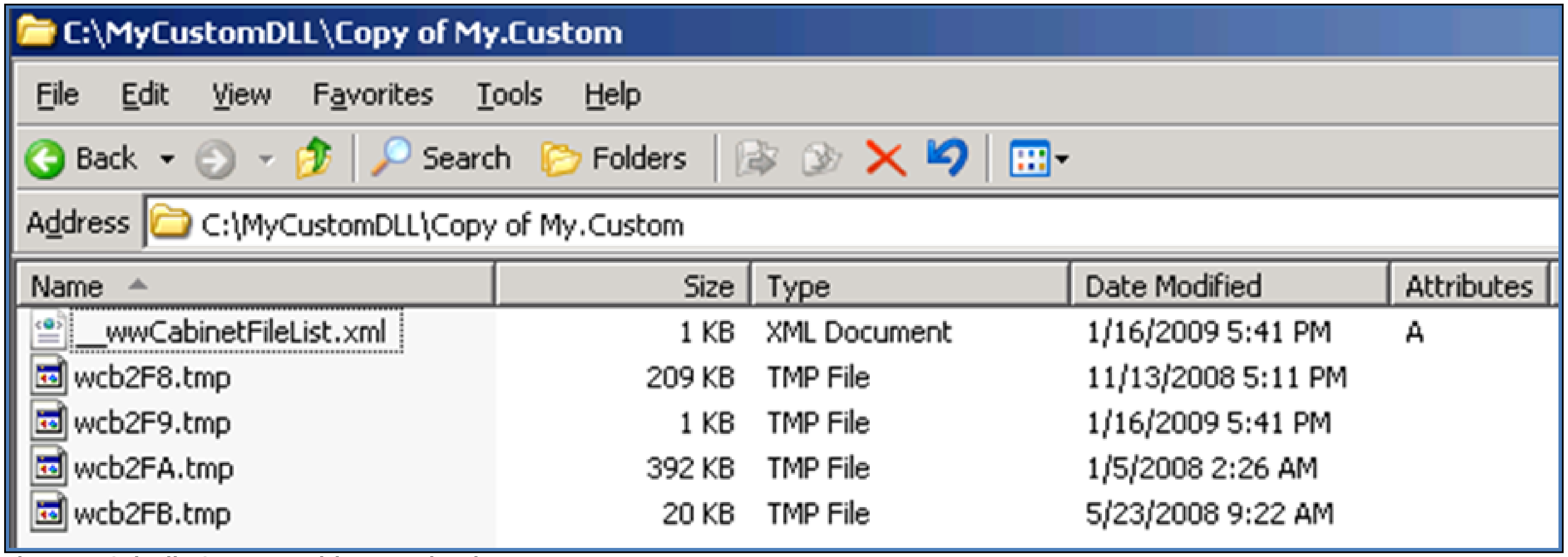
Figure 6 (below) shows the contents of the __**wwCabinetFileList.xml** file.



```
- <WWCabinetFileList>
    <FileItem FileName="My.Custom.dll" FileCode="wcb2EF.tmp" />
    <FileItem FileName="My.Custom.xml" FileCode="wcb2F0.tmp" />
  </WWCabinetFileList>
```

Figure 6: Contents of the XML File

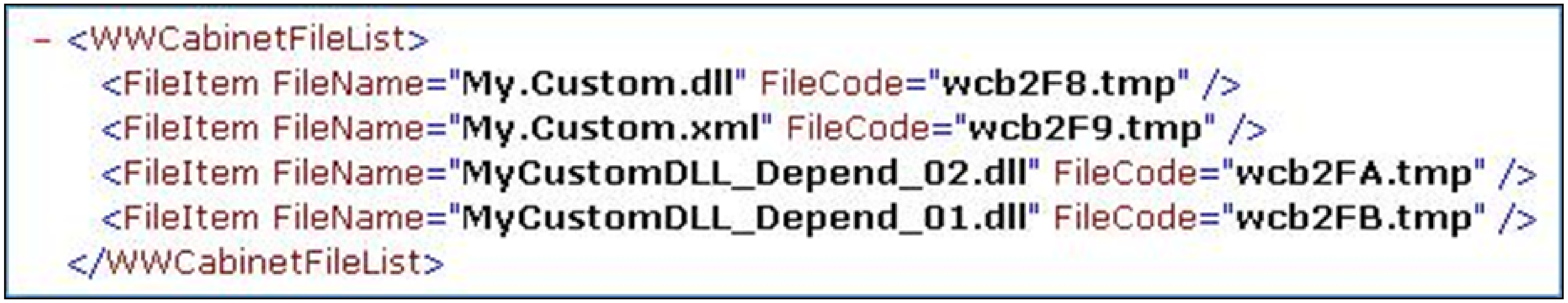
Figure 7 (below) shows the the contents of the extracted cab file for My.Custom.aaSLIB file (with dependencies).


C:\MyCustomDLL\Copy of My.Custom
File Edit View Favorites Tools Help
Back Search Folders
Address C:\MyCustomDLL\Copy of My.Custom

Name	Size	Type	Date Modified	Attributes
__wwCabinetFileList.xml	1 KB	XML Document	1/16/2009 5:41 PM	A
wcb2F8.tmp	209 KB	TMP File	11/13/2008 5:11 PM	
wcb2F9.tmp	1 KB	TMP File	1/16/2009 5:41 PM	
wcb2FA.tmp	392 KB	TMP File	1/5/2008 2:26 AM	
wcb2FB.tmp	20 KB	TMP File	5/23/2008 9:22 AM	

Figure 7: Cab File Contents with Dependencies

Figure 8 (below) shows the contents of the __**wwCabinetFileList.xml** file.



```
- <WWCabinetFileList>
    <FileItem FileName="My.Custom.dll" FileCode="wcb2F8.tmp" />
    <FileItem FileName="My.Custom.xml" FileCode="wcb2F9.tmp" />
    <FileItem FileName="MyCustomDLL_Depend_02.dll" FileCode="wcb2FA.tmp" />
    <FileItem FileName="MyCustomDLL_Depend_01.dll" FileCode="wcb2FB.tmp" />
  </WWCabinetFileList>
```

Figure 8: XML File with Dependencies

Loading Script Function Libraries into Memory

Application Server loads into memory the required Script Function Libraries when the object's hosting engine is on-scan.

This means that even if a modified version of the custom DLL were to be imported into the Galaxy, and even if the objects that reference the script function library were undeployed and redeployed, the existing DLL *already loaded into memory* and associated to the **aaEngine.exe** process is still the one referenced by object scripts.

You must unload the existing DLL from memory in order to ensure the most recent version of the DLL is being used.

Do this by **Shutting Down** the engine. Use the SMC Platform Manager and then set the engine On-scan again.

Undeploying and redeploying the entire hosting Platform achieves the same effect.

References

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