Creating Client/Server Applications: Managing clients, Managing Plug-ins, tips and tricks

By Chiheb NASR, QA Engineer, 4D S.A.

Technical Note 06-21

Introduction

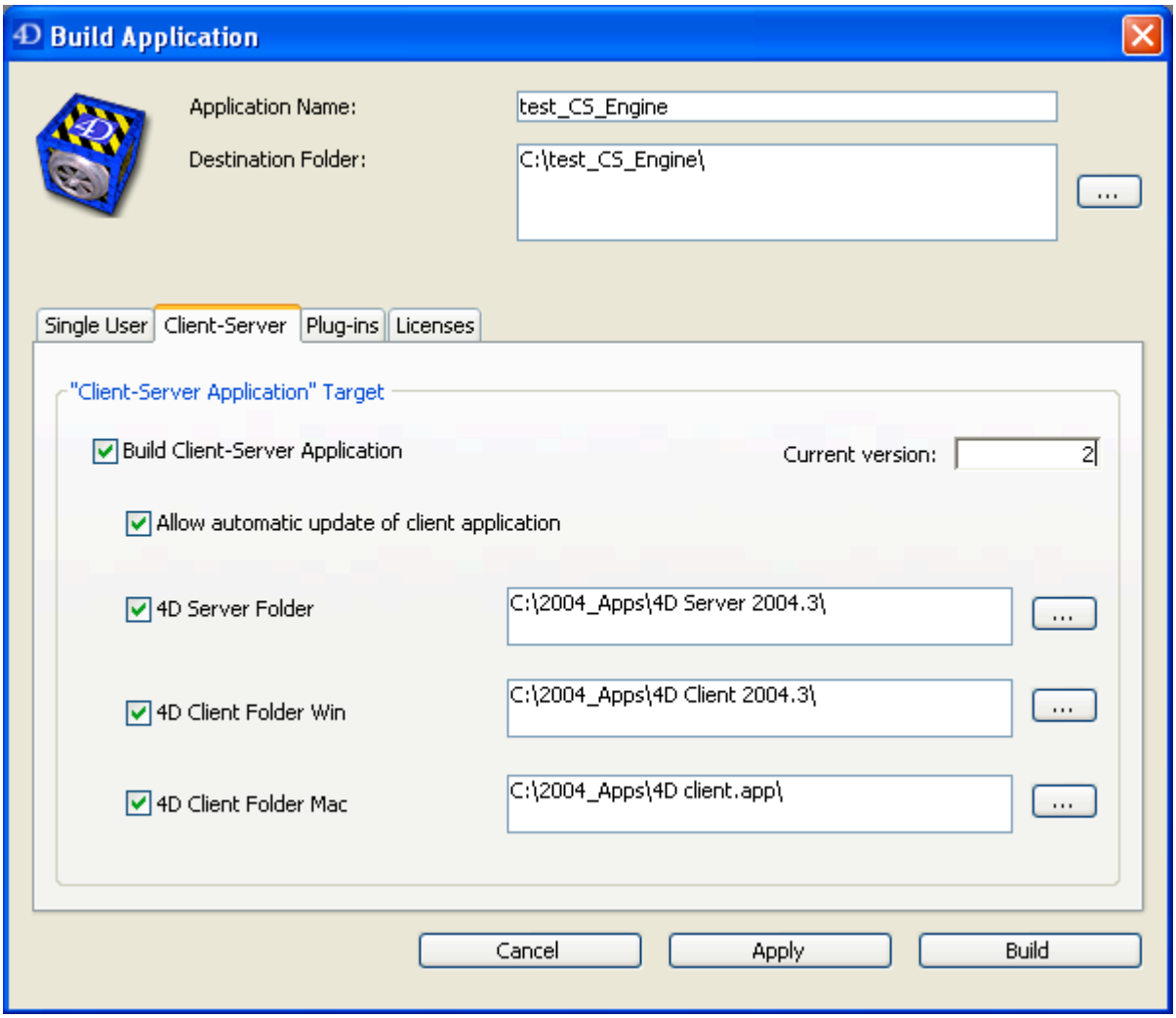
4D 2004 introduces a new custom client/server multiplatform application generator that allows an automatic update of the client. This build feature can either be accessed though the Build application dialog in the Design environment or through an XML project file. Using the XML project file provides many more possibilities. The purpose of this technical note is to explain the mechanisms that come into play with this application generator and to demonstrate its use.

Prerequisites:

It is highly recommended to read the documentation on the Build application command as well as technical note 33569 (http://www.4d.com/knowledgebase?CaseID=33569).

Building multiplatform client applications

The Client-Server page of the Build Application dialog allows you to select the Mac and Windows clients that will be used to create the client applications. When you select the optionD Client Folder Win, you must select the path to the folder that contains the Windows 4D Client application. On the other hand, when selecting the 4D Client Folder Mac, you will have to select the path to the 4D Client package (4D Client.app) and not the path to the folder that contains the package. This applies to both platforms. Below is an example of a Client/Server setup:



You cannot generate a double-clickable client application outside the platform it will run on. On Windows you can only build the Windows client and the same restriction applies to the Mac platform. In addition to the license restrictions (you have to own the license of a Developer Edition for each platform you want to deploy a customized client to), neither OS can handle the specifics of the executable of the other platform.

After the compilation and generation of the 4D Client Engine (executable), the file, you will have to create the file "EnginedServer.xml" and place it in the "4D Extensions" folder. This file allows you to automatically connect to the server, without having to enter the IP address or the port number. If you create the client through the Build Application dialog, you cannot specify the IP address of the server nor the port number the server will be using. In this case, the server's IP address that will be used is the address of the machine the build was performed on. You can work around this by simply creating the EnginedServer.xml file. Here is an example of the contents of this file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

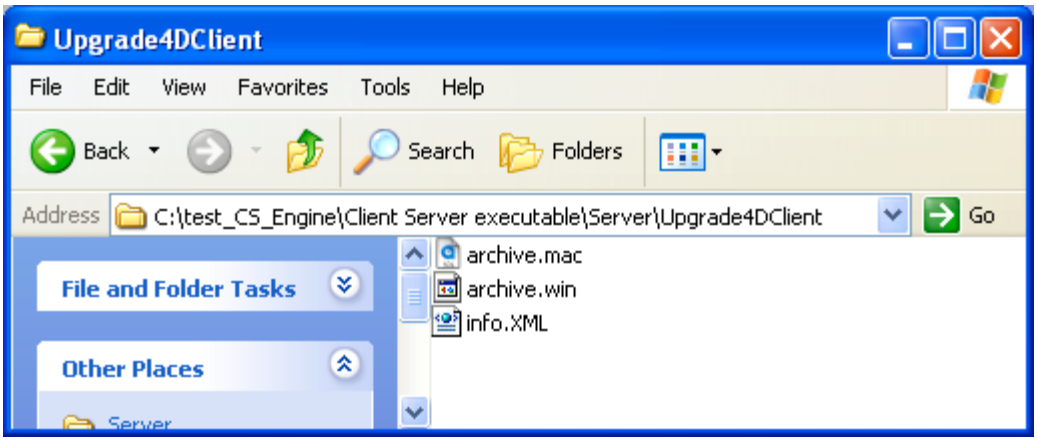
```
<Connection>
   <IPAdress>192.68.91.10,19815</IPAdress>
</Connection>
```

The port number is optional. If it is not specified, the default port number will be used (*19813*).

The absence of the "EnginedServer.xml" file when the build is performed through the Build application dialog has a primary advantage: since it would be created at the time the application is built, it would just insert the build machine's IP address. When the client is first started and the EnginedServer.xml file is missing, the client tries to find the server. If it does, an EnginedServer.xml file is created. If it does not, the client displays the manual connection dialog. When using an XML project file to create the build through the BUILD APPLICATION command, you can use the XML keys to define what is to be inserted in the EnginedServer.xml file. The example below demonstrates the some of the keys to insert in the build project file (BuilApp.XML):

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Preferences4D>
      <BuildApp>
         <CS>
            <BuildServerApplication>True</BuildServerApplication>
             <BuildCSUpgradeable>True</BuildCSUpgradeable>
             <CurrentVers>2</CurrentVers>
             <HardLink>Mabase_CS_Engine</HardLink>
             <IPAddress>192.68.10.10</IPAddress>
            <PortNumber>19815</ PortNumber >
          </CS>
       </BuildApp>
 </Preferences4D>
```

Also, when you check the options 4D Client Folder Win and 4D Client Folder Mac, a folder named Upgrade4DClient is created that includes the client archives (archive.mac and archive.win). This is true regardless of the platform you are building the application on. Those archives are used for the automatic update of the client applications. The screenshot below displays the typical contents of the Upgrade4DClient folder.



Note:
A customized Client/Server application runs only with a compiled database. This implies that you will not be able to use the features described in this technical note with a server that runs in interpreted mode or within a client server development environment.

Using the XML keys to customize your Client/Server applications

The use of the XML keys allows you to further customize your application and to automate the creation of new versions destined to be deployed.

We will go through an example of the creation of a client/server application. To use this project, you will have to call the BUILD APPLICATION command and pass the path to the XML project as a parameter. The BUILD APPLICATION command automatically launches a compilation if the database is not compiled.

This example uses several XML keys to handle the following aspects:

- IP address of the server ;
- Port number of the server (if different from 19813);
- Specific icons for 4D Server Windows and 4D clients (Mac et Windows),
- Version management (minimum version, maximum version and current version),
- Plug-in management.

Note:
Please note that the following tag names were changed since version 2004.1. The string « *FolderIsValid* » was replaced by « ***IncludeIt*** » for clarity purposes.

/Preferences4D/BuildApp/SourcesFiles/RuntimeVL/RuntimeVLFolderIsValid
/Preferences4D/BuildApp/SourcesFiles/CS/ServerFolderIsValid
/Preferences4D/BuildApp/SourcesFiles/CS/ClientWinFolderIsValid
/Preferences4D/BuildApp/SourcesFiles/CS/ClientMacFolderIsValid

The new tags now are:

*/Preferences4D/BuildApp/SourcesFiles/RuntimeVL/**RuntimeVLIncludeIt***
*/Preferences4D/BuildApp/SourcesFiles/CS/**ServerIncludeIt***
*/Preferences4D/BuildApp/SourcesFiles/CS/**ClientWinIncludeIt***
*/Preferences4D/BuildApp/SourcesFiles/CS/**ClientMacIncludeIt***

Note:
We recommend renaming your projects and placing them in a location different from the default location (\Preferences\BuildApp), otherwise they will be overridden when you use the Build Application dialog.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Preferences4D>
	<BuildApp>
		<BuildCompiled>False</BuildCompiled>
		<BuildApplicationSerialized>False</BuildApplicationSerialized>
		<BuildApplicationLight>False</BuildApplicationLight>
	<SourcesFiles>
		<RuntimeVL>
			<RuntimeVLIncludeIt>False</RuntimeVLIncludeIt>
		</RuntimeVL>
		<CS>
			<ServerIncludeIt>True</ServerIncludeIt>
			<ClientWinIncludeIt>True</ClientWinIncludeIt>
			<ClientMacIncludeIt>True</ClientMacIncludeIt>
			<ServerWinFolder>F:\PC\4D Server\</ServerWinFolder>
			<ClientWinFolderToWin>F:\PC\4D Client\</ClientWinFolderToWin>
			<ClientMacFolderToWin>F:\Mac\4D Client.app\</ClientMacFolderToWin>
			<ServerIconWinPath>F:\PC\pc\Server.ico</ServerIconWinPath>
			<ClientWinIconForWinPath>F:\PC\pc\Client.ico</ClientWinIconForWinPath>
			<ClientMacIconForWinPath>F:\PC\Mac\client.icns</ClientMacIconForWinPath>
		</CS>
	</SourcesFiles>
	<BuildApplicationName>Mabase_CS_Engine</BuildApplicationName>
	<BuildWinDestFolder>F:\TEST_CS_ENGINE\Application_Cible\</BuildWinDestFolder>
	<CS>
		<BuildServerApplication>True</BuildServerApplication>
		<BuildCSUpgradeable>True</BuildCSUpgradeable>
		<RangeVersMin>1</RangeVersMin>
		<RangeVersMax>5</RangeVersMax>
		<CurrentVers>2</CurrentVers>
		<HardLink> Mabase_CS_Engine </HardLink>
		<IPAddress>192.68.10.10</IPAddress>
		<PortNumber>19815</ PortNumber >
	</CS>
	<ArrayExcludedPluginName>
		<ItemsCount>1</ItemsCount>
		<Item1>4D View</Item1>
	</ArrayExcludedPluginName>
```

```
        <ArrayExcludedPluginID>
          <ItemsCount>1</ItemsCount>
          <Item1>13000</Item1>
        </ArrayExcludedPluginID>
        <Licenses>
          <ArrayLicenseWin>
            <ItemsCount>1</ItemsCount>
              <Item1>E:\Documents and Settings\All Users\Application
                Data\4D\Licenses\4SDE80….html</Item1>
        </ArrayLicenseWin>
        <ArrayLicenseTarget>
          <ItemsCount>1</ItemsCount>
          <Item1>1</Item1>
        </ArrayLicenseTarget>
        </Licenses>
      </BuildApp>
</Preferences4D>
```

4D Client Update

The option « Allow automatic update of the client application» in the Build Application dialog triggers the update of all the clients the first time they connect after a new version of the server was installed. You can also use the XML key **<BuildCSUpgradeable>** in the XML project file to enable (True) or disable (False) that feature.

For example, the following XML key enables the option:

<BuildCSUpgradeable>**True**</BuildCSUpgradeable>.

The upgradable client feature allows you to make sure the versions between 4D Client and 4D Server match and renders obsolete the need to manually perform upgrades on each client machine. During the automatic upgrade, the server sends its updated archive to the client. Once the archive is received, the client builds its successor, deletes itself, and connects to the server.

The update mechanism deserves to be fully understood to face any possible problems that may arise. The beginning of the update process is handled by the 'old' 4D Client, and then a script takes over (« upgclnt.bat » on Windows or « upgclnt.sh » on Mac OS) to complete the process. These scripts can be found in the "4D Extensions" folder of the new 4D Client.

In the following table we list the all the actions that are taken during the upgrade:

Step	Description	Notes
1	The old 4D Client sends a request to the server to know what versions are compatible with the server: current, minimum and maximum version.	
2	The old 4D Client checks to see if its version matches the server's version or is within the acceptable range	
3	If the version is compatible, then the connection is established, otherwise an update confirmation dialog is displayed.	The download uses the same mechanism as the copy of the local resources.
4	If the dialog is confirmed, the download of the new archive begins, otherwise 4D Client quits.	
5	The archive is expanded in a temporary folder named $temp4dclient created at the same level as the old 4D Client.	The old client performs the expanding.
	The scripts upgclnt.bat or upgclnt.sh are copied to the temporary folder of the current session. The scripts are located in the 4D Extensions folder of the new client.	Once copied, the scripts are deleted.
6	The old 4D Client launches the script that matches its platform (upgclnt.bat or upgclnt.sh).	There is a delay in the script to accommodate for 4D Client quitting.
7	The old client quits.	
8	On the Mac, the script upgclnt.sh makes the 4D Client package executable in the current session (using the Unix chmod command).	Mac OS only.
9	The script upgclnt.bat or upgclnt.sh retrieves the new 4D Client from the $temp4DClient folder and places the new client in the same folder as the old client.	
10	The script upgclnt.bat or upgclnt.sh deletes the $temp4DClient folder.	
11	The script upgclnt.bat or upgclnt.sh deletes the old client.	On Windows, the old 4D Client is placed in a temporary folder named $Trash4D before this folder is placed in the trash.
12	The script upgclnt.bat or upgclnt.sh launches the new client, which automatically connects to the server.	

How are the plug-ins generated in Client/Server?

The Build application dialog allows you to designate each plug-in that you want to see integrated to the application. By default, all plug-ins loaded through 4D Stand-alone

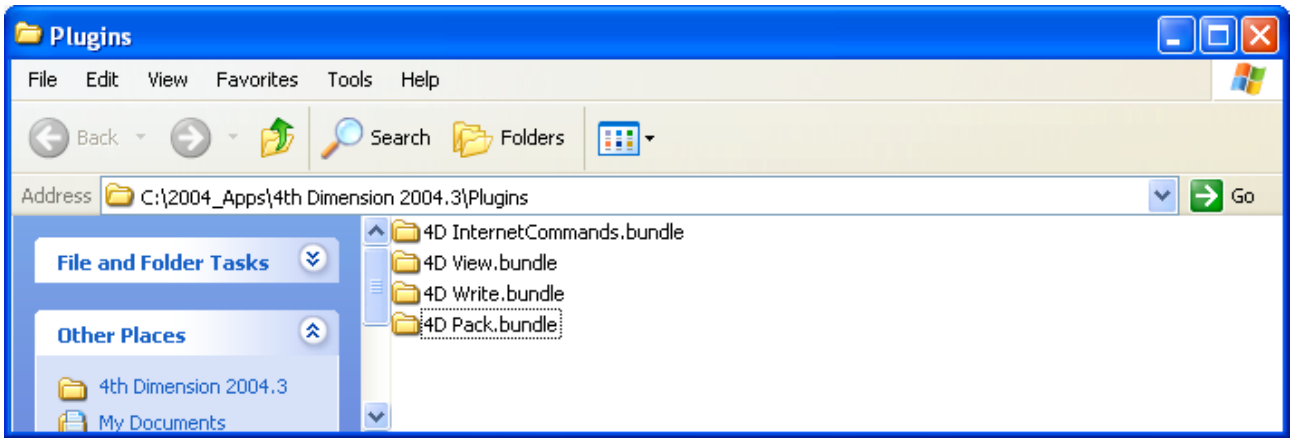
are included. It is up to the developer to actually select or deselect plug-ins in the third page of the Build application dialog or to specify it in the XML project using the keys <ArrayExcludedPluginName> and <ArrayExcludedPluginID>. In the previous XML project, you can see that only the plug-in ID 13000 will be excluded from the build.

To integrate plug-ins to the Client/Server application generated, you can place them in three different locations:

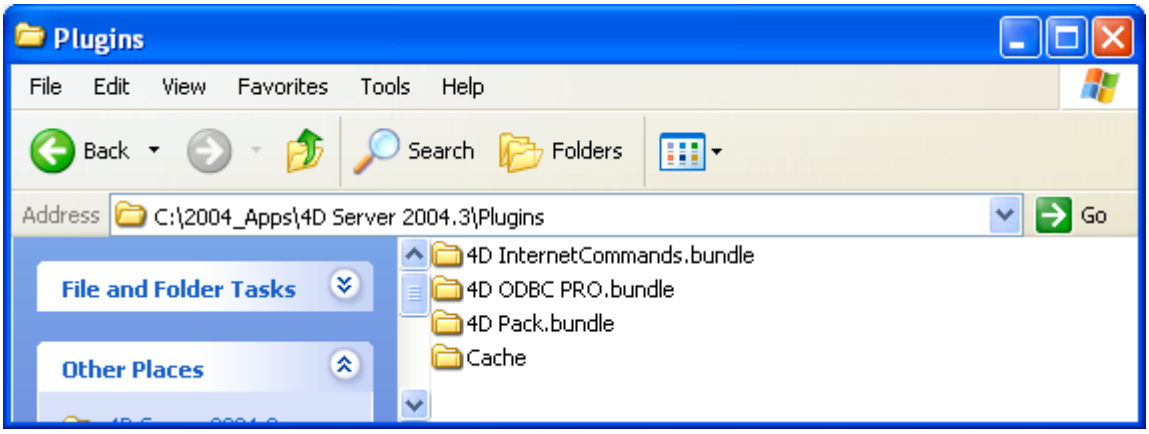
- In 4D's 'Plugins' folder that is used for the compilation and the build;
- In the 'plugins', 'Mac4dx'or 'Win4dx' folder of the source database;
- In the 'plugins' folder of 4D Server.

Note:
In case of conflict between two different versions of the same plug-in, the priority always goes to the 'Plugins' folder of 4D Server.

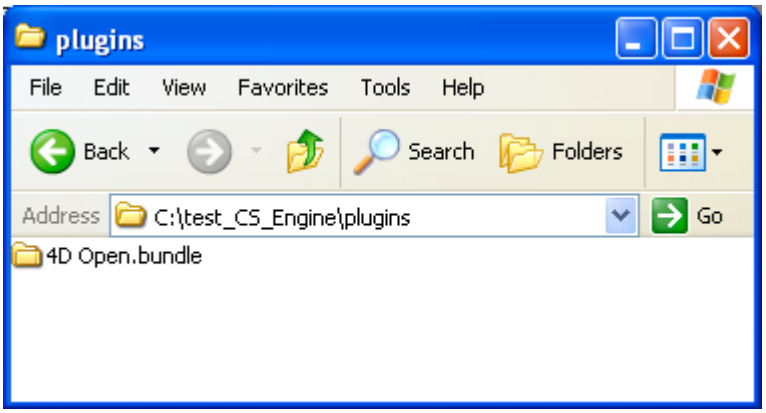
Let's run a fictitious scenario that will highlight the integration mechanisms. Let's assume we place the following plug-ins (**A**) in the 'plugins' folder of the 4D Stand alone application that is used to create the client/server application:



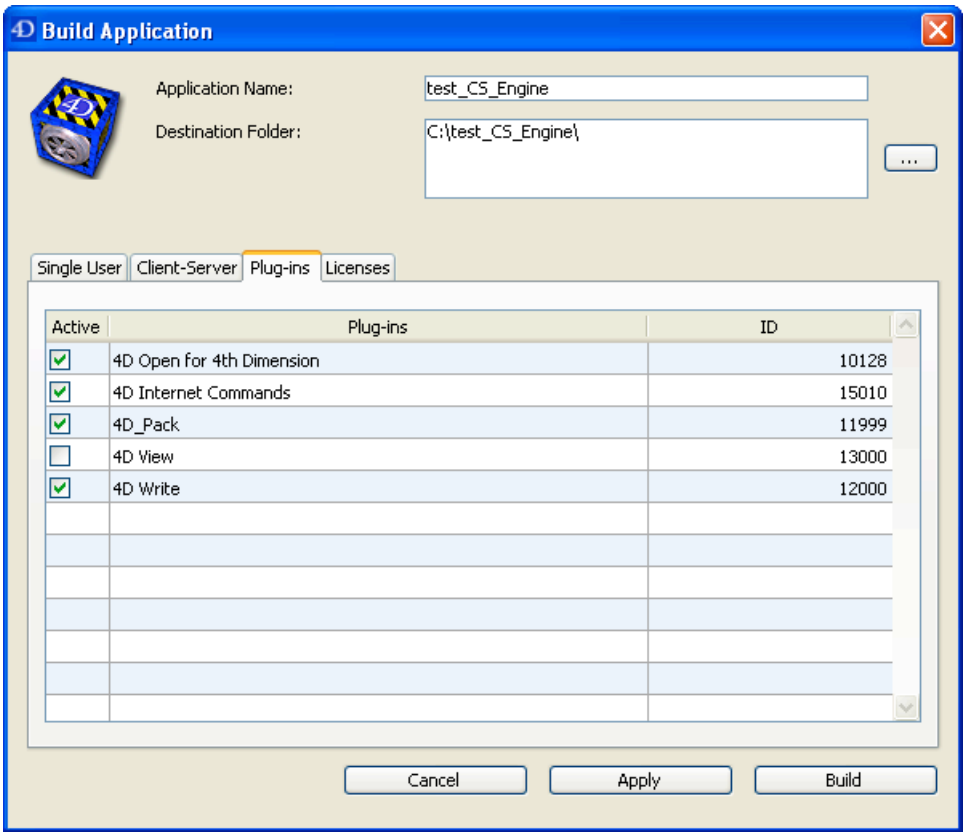
Then, let's assume that the 'plugins' folder of 4D Server contains the following plug-ins (**B**):



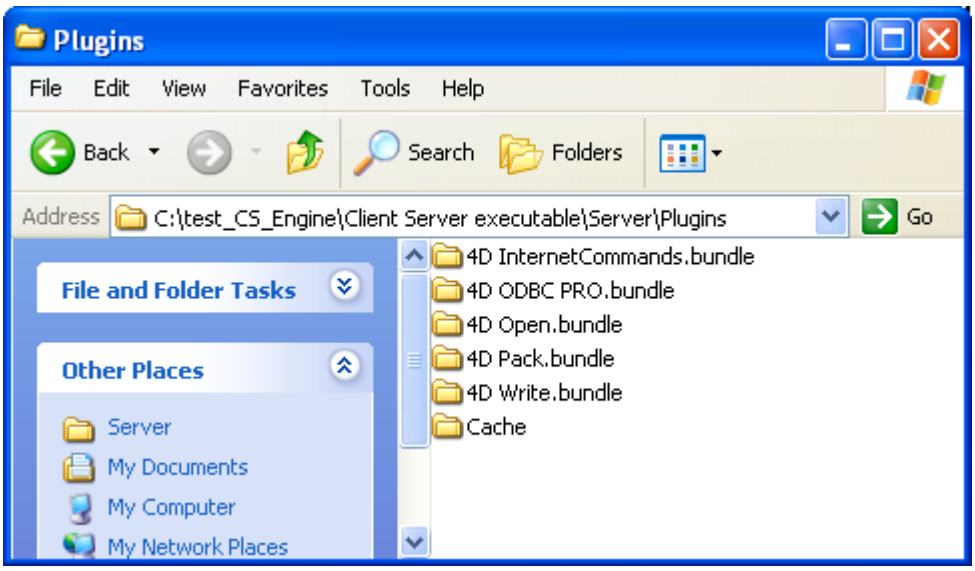
Last, the 'plugins' folder of the source database contains the 4D Open for 4D plug-in(**C**):



Obviously, there will only be (**A**) and (**C**) plug-ins that will be loaded in the development environment, as illustrated in the screenshot below:



Note that the 4D View plug-in was deselected so that it could not be present in the build of the application. After compiling and building the application, here are the contents of the plugins folder of the final 4D Server built:



You can note the presence of all the plug-ins (**A**), (**B**) and (**C**), at the exception of 4D View that was voluntarily deselected before the build. You can also note that the plug-ins Internetcommands.bundle and 4D Pack.bundle come from the plugins folder of 4D Server, because of the priority mentioned earlier. Also, this scenario implies the use of the following XML key (to exclude 4D View):

```
<ArrayExcludedPluginName>
    <ItemsCount>1</ItemsCount>
    <Item1>4D View</Item1>
</ArrayExcludedPluginName>

<ArrayExcludedPluginID>
    <ItemsCount>1</ItemsCount>
    <Item1>13000</Item1>
</ArrayExcludedPluginID>
```

Note:
If a plug-in is placed twice under different names, the ID conflict will be detected when 4D Server is launched. The alert that is displayed lets you know that the plug-in is installed twice and 4D Server quits.

Conclusion

In this technical note, we detailed some mechanisms used during the build of client/server applications. It covers the management of plug-ins as well as the mechanism the automatic upgrade.