Kepware Technologies

Creating a ClientAce Service Application

January, 2013
Ref. 1.02

Table of Contents

1. Overview....1
2. Creating a Service....1
   - 2.1 Creating and Configuring the Service....1
3. Adding the Reference for the ClientAce API....1
   - 3.1 Adding the Kepware.ClientAce API to the Project....1
     - 3.1.1 Visual Basic .Net (VB .Net)....1
     - 3.1.2 Visual C# (C#)....1
4. Setting the Application to Target the .Net 4.0 Framework....2
5. Disabling the Visual Studio Hosting Process....2
6. Adding Features to the Service....2
   - 6.1 Adding Initialization Code....2
   - 6.2 Adding Code to Run when the Service Starts....3
   - 6.3 Adding Code to Run when the Service Stops....4
7. Creating Installers for the Service....4
8. Building the Service Project....5
9. Installing the Project as a Service....5
10. Starting and Stopping the Service....5
11. Uninstalling the Service....6
12. Summary....6

1. Overview

This document intends to demonstrate the steps for both creating a simple Windows Service application and making a simple connection to an OPC Server. The basic steps required are as follows:

- Create a project by using the Windows Service application template. This template creates a class that inherits from ServiceBase and writes much of the basic service code (such as the code to start the service).
- Write the code for the OnStart and OnStop procedures, and override any other methods that need to be redefined.
- Add the necessary installers for the service application. A class that contains two or more installers will be added to the application by default when the Add Installer link is selected. One installs the process, and the other is used for each associated service that the project contains.
- Build the project.
- Create a setup project to install the service, and then do so.
- Access the Windows Services Control Manager and start the service.

Note: This example is written in Visual Studio 2010 and uses ClientAce V3.5.0.9. Because it targets the .Net 4.0 frameworks, support must be added in order to use the .Net 3.5 Binaries.

2. Creating a Service

To begin, users must create the project and then set the values that are required for the service to function correctly.

2.1 Creating and Configuring the Service

1. In the **File** menu, click **New Project**.
2. In the list of Visual Basic or Visual C# project templates, locate and select **Windows Service**. Name the project "MyServiceApp". Then, click **OK**.
3. Next, click on the designer to select **Service1**.
4. In **Properties**, set both **ServiceName** and **(Name)** to "MyServiceApp".

3. Adding the Reference for the ClientAce API

3.1 Adding the Kepware.ClientAce API to the Project

3.1.1 Visual Basic .Net (VB .Net)

1. In VB.Net, beneath **Solution Explorer**, right-click on the project and then select **Add Reference**.
2. Next, open the **.Net** tab.
3. Browse the list for "Kepware.ClientAce". Then, double-click to accept it.

3.1.2 Visual C# (C#)

1. In C#, beneath **Solution Explorer**, right-click on the project and then select **Add Reference**.
2. Next, open the **.Net** tab.

3. Browse the list for "Kepware.ClientAce". Then, double-click to accept it.

4. Setting the Application to Target the .Net 4.0 Framework

ClientAce's V3.5.0.x binaries target the .Net 3.5 Framework; as such, the easiest way to use it in Visual Studio 2010 is to set the target .Net Framework to .Net 3.5. Because ClientAce may need to be used in projects whose components require .Net 4.0 support, it can also be configured to use .Net 4.0 for all components except the ClientAce DA Junction. For more information, refer to the instructions below.

1. To start, add an **Application Configuration** file. To do so, right-click on the project in **Solution Explorer** and then select **Add** | **New Item**.
2. Next, locate the application configuration file beneath **General**. Then, add the following code:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<startup useLegacyV2RuntimeActivationPolicy="true">
<supportedRuntime version="v2.0"/>
<supportedRuntime version="v4.0"/>
</startup>
</configuration>
```

3. Next, set the target .Net Framework to **.Net Framework 4 Client Profile**.

   Note: In C#, the target may be located by clicking **Project Properties** | **Application**. In VB.Net, the target may be located by clicking **Project Properties** | **Compile** | **Advanced Compile Options**.

5. Disabling the Visual Studio Hosting Process

The hosting environment is used for debugging. When enabled, it forces the service to run with incorrect permissions. For information on disabling it, refer to the instructions below.

1. In VB.Net or C#, open **Project Properties** and then select the **Debug** tab.
2. Next, uncheck **Enable the Visual Studio hosting process**.

6. Adding Features to the Service

Procedures and features may be added that can be used while running.

6.1 Adding Initialization Code

1. In **Solution Explorer**, right-click on **Service1.vb** or **Service1.cs**. Then, select **View Designer**.
2. Next, drag the desired components from the Toolbox. Use the **Properties** window for configuration.

   Note: When running as a service application, there will be no interaction with the desktop. Components that create popups or require direct action from the user should not be selected.
3. Next, right-click on **Service1.vb** or **Service1.cs** and select **View Code.**
4. To initialize the OPC Objects, add the code displayed below.

VB

```
Public Class Service1
    Dim WithEvents Server As New Kepware.ClientAce.OpcDaClient.DaServerMgt
    Dim connectInfo As New Kepware.ClientAce.OpcDaClient.ConnectInfo
    Dim connectFailed As Boolean
    Dim clientHandle As Integer
    Dim url As String
End Class
```

C#

```
public partial class Service1 : ServiceBase
{
    //Initilize the server object
    Kepware.ClientAce.OpcDaClient.DaServerMgt DAserver = new
Kepware.ClientAce.OpcDaClient.DaServerMgt();

    //Initilize the connection info class
    Kepware.ClientAce.OpcDaClient.ConnectInfo connectInfo = new
Kepware.ClientAce.OpcDaClient.ConnectInfo();
    bool connectFailed;
    public Service1()
    {
        InitializeComponent();
    }
}
```

6.2 Adding Code to Run when the Service Starts

1. In the **Code Editor**, locate the **OnStart** method that was automatically overridden when the project was created.
2. Then, write the code to determine what will occur when the service starts running (such as connecting to the OPC Server).

VB

```
Protected Overrides Sub OnStart(ByVal args() As String)
        ' Add code here to start your service. This method should set things
        ' in motion so your service can do its work.

        'Initialize the new server object
        Server = New Kepware.ClientAce.OpcDaClient.DaServerMgt
        'Specify the server client handle
        clientHandle = 1
        'Initialize the connect info object data
        connectInfo.LocalId = "en"
        connectInfo.KeepAliveTime = 5000
        connectInfo.RetryAfterConnectionError = True
        connectInfo.RetryInitialConnection = False
        connectFailed = False
        'Set the url for the connection to the V5 server
        url = "opcda://localhost/Kepware.KEPServerEX.V5"
        Try
            'Attempt to connect to the server
            Server.Connect(url, clientHandle, connectInfo, connectFailed)

        Catch ex As Exception

        End Try
    End Sub
```

C#

```
protected override void OnStart(string[] args)
        {
            // Initialize the connect info object data
            connectInfo.LocalId = "en";
            connectInfo.KeepAliveTime = 5000;
            connectInfo.RetryAfterConnectionError = true;

            //Try the server connection
            DAserver.Connect("opcda://localhost/Kepware.KEPServerEX.V5", 0, ref connectInfo, out
connectFailed);

        }
```

6.3 Adding Code to Run when the Service Stops

1. In the **Code Editor**, locate the **OnStop** method that was automatically overridden when the project was created.
2. Then, write the code to determine what will occur when the service stops running (such as when disconnecting from the OPC Server).

VB

```
    Protected Overrides Sub OnStop()
        ' Add code here to perform any tear-down necessary to stop your service.
        'Disconnect from server
        If Server.IsConnected Then
            Server.Disconnect()
        End If
    End Sub
```

C#

```
        protected override void OnStop()
        {
            //Disconnect from the server
            if (DAserver.IsConnected)
            DAserver.Disconnect();

        }
```

7.Creating Installers for the Service

1. In **Solution Explorer**, right-click on **Service1.vb** or **Service1.cs**. Then, select **View Designer**.
2. Next, click in the background of the designer to select the service instead of its contents. With the designer in focus, right-click and select **Add Installer**.

   Note: A component class that contains two installers will be added to the project by default. The component is named "ProjectInstaller," and it contains installers for both the service and for the service's associated process.

3. In **ProjectInstaller**, open the **Design** view. For a VB project, click **ServiceInstaller1**. For a C# project, click **serviceInstaller1**.
4. In the **Properties** window, make sure that the **ServiceName** property is set to **MyServiceApp**.
5. Then, set the **StartType** property to **Automatic**.

6. In the designer, click **ServiceProcessInstaller1** for a VB .Net project, or **serviceProcessInstaller1** for a C# project. Then, set the **Account** property to **LocalSystem** to make the service install and run on a local service account.

   Note: The LocalSystem account has broad permissions—including the ability to write to the Event Log. This is the same account that KEPServerEX runs under by default when running as a service. It allows connectivity with minimal changes to DCOM.

8. Building the Service Project

1. To start, sign the project so that it does not revert to Demo Mode.
2. In **Solution Explorer**, right-click on the project and select **Properties**.
3. In the **Property Designer**, open the **Application** page.
4. In the **Startup Object** list, click **MyServiceApp.Program**.
5. Then, press **Ctrl**+**Shift**+**B** to build the project.

   Note: Now that the Windows service is built, it can be installed. To do so, users must have administrative permissions on the computer on which it is being installed.

9. Installing the Project as a Service

1. In the **Start** menu, open the shortcut for **Command Prompt** and select **Run As Administrator**.
2. Next, navigate to the folder that contains the project's output. For example, in the **My Documents** folder, navigate to *Visual Studio 2012\Projects\MyServiceApp\bin\Debug*.
3. Then, enter the following command:

   ```
   InstallUtil.exe MyServiceApp.exe
   ```

   Note: If the service installs successfully, the "InstallUtil.exe" file will report success.

   Important: If more than one .Net development environment is installed, the operating system will not be able to find the "InstallUtil.exe" file and the exact location will need to be specified. The "InstallUtil.exe" file used must be for the application's target .Net Framework. The examples below use .Net 4.0. A .BAT file will be created to run from the command prompt using the command line displayed below.

   ```
   C:\Windows\Microsoft.NET\Framework\v4.0.30319\installutil.exe MyServiceApp.exe
   ```

10. Starting and Stopping the Service

1. Open the **Start Menu**. To access the **Services Control Manager** in Windows 7, Windows Vista, and Windows Server, right-click on **Computer** and then select **Manage**.
2. In the **Computer Management** console, expand **Services and Applications** (located in the left pane).
3. Then, click **Services**. The **MyServiceApp** should be displayed beneath it.
4. Right-click on the service and select **Start**.

Note: A connection to the server should be visible.

5. When ready, right-click on the service and select **Stop**.
   Note: The connection should be removed from the server.

11. Uninstalling the Service

1. In the **Start** menu, open the shortcut for **Command Prompt** and then select **Run As Administrator**.
2. Next, navigate to the folder that contains the project's output. For example, in the **My Documents** folder, navigate to *Visual Studio 2012\Projects\MyServiceApp\bin\Debug*.
3. Then, enter the following command:

```
InstallUtil.exe /u MyServiceApp.exe
```

Note: If successful, the "InstallUtil.exe" file will report that the service was removed.

Important: If more than one .Net development environment is installed, the operating system will not be able to find the "InstallUtil.exe" file and the exact location will need to be specified. The "InstallUtil.exe" file used must be for the application's target .Net Framework. The examples below use .Net 4.0. A .BAT file will be created to run from the command prompt using the command line displayed below.

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\installutil.exe /u MyServiceApp.exe
```

12. Summary

At this time, users should be able to create a ClientAce service application.