



Creating Monarch Applications from Scratch for ASNA Monarch® 4.0



Creating Monarch Applications from Scratch

Information in this document is subject to change without notice. Names and data used in examples are fictitious unless otherwise noted.

No component of **Creating Monarch Applications From Scratch** may be reproduced, disassembled, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form without the written permission of ASNA (Amalgamated Software of North America).

Copyright © 2006-2007 ASNA - Amalgamated Software of North America. All rights reserved.

April 17, 2008

Contents

Chapter 1 – Introduction	1
Chapter 2 – Creating the Presentation Layer	3
Preparing a Website.....	3
Adding Display Files.....	6
Completing the Web Site.....	9
Chapter 3 – Creating the Business Logic Layer	13
Creating a Class Library Project.....	13
Adding a Program to the Library.....	14
Making <i>Translate</i> Callable	15
Doing the Real Thing.....	16
Getting the Ball Rolling.....	17
Connecting Back to the Web Site.....	19
Appendix	21
Adding Monarch Controls to VS 2005 Toolbox	21

Chapter 1 – Introduction

As part of ASNA Monarch, the Monarch Framework is provided to facilitate the migration of iSeries applications. The Monarch Framework is comprised of a set of classes and a particular style of application creation. An application that follows this style and uses these classes is termed a “Monarch Application”.

The Monarch Framework borrows many of its concepts and terminology from OS/400. The **ASNA Monarch Application Architecture Concepts** manual explains these concepts in depth, and this manual is a **required** reading for anyone trying to create or maintain Monarch applications.

The Framework components are divided into two groups; the first one deals with providing an **Execution Context** for the application, while the second provides **Display File** support for interactive applications.

This document describes the process of creating a Monarch application from scratch. Parts of this process can also be used when there is a need of increasing the capabilities of an existing application.

Applications are divided into two big groups: **Interactive** and **Non-Interactive**¹. Batch applications are simpler, so we will concentrate on an **Interactive** application in this document. The general steps to creating an interactive application are:

1. Prepare a web site to host the presentation layer.
2. Add display files to the site.
3. Create a Class Library project to implement the business logic of the application.
4. Add programs to the class library.

Chapter 2 addresses steps 1 and 2. Chapter 3 addresses steps 3 and 4.

In the examples that follow throughout, we will refer to an application called **MyApp**. The completed application will be a translator of phrases written in any earthly language into the language of cats (*Felis silvestris catus*).

Once you have completed this exercise, you will have the knowledge of how an interactive Monarch application is constructed.

This document uses Monarch 4.0 and Visual Studio 2005. Future releases of this document will expand to include the creation of **Non-Interactive** Monarch applications.

¹ Non-Interactive is further divided into **Process** or **Service** programs (.dll) and **Batch** programs (.exe).

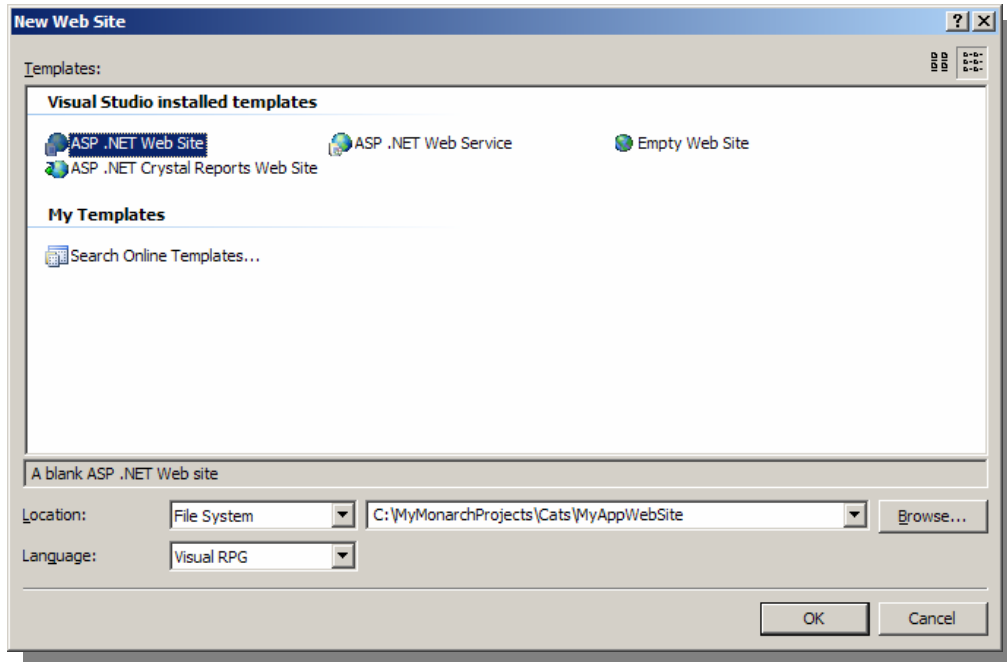
This page intentionally left blank.

Chapter 2 – Creating the Presentation Layer

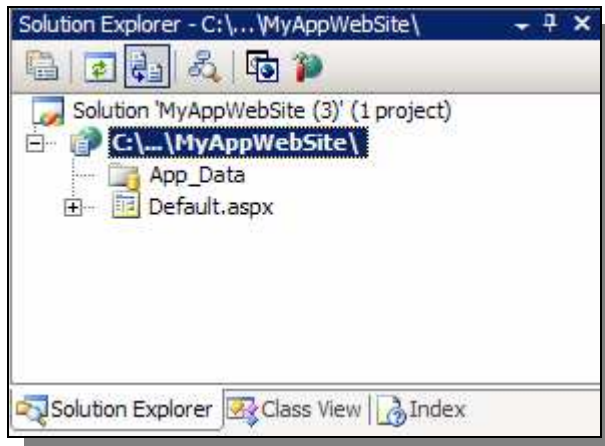
The presentation layer will be hosted on a Web site.

Preparing a Website

1. Using Visual Studio, create a new **ASP.NET Web Site** (File- New - Web Site). In this illustration, the name of the web site is **MyAppWebSite**.



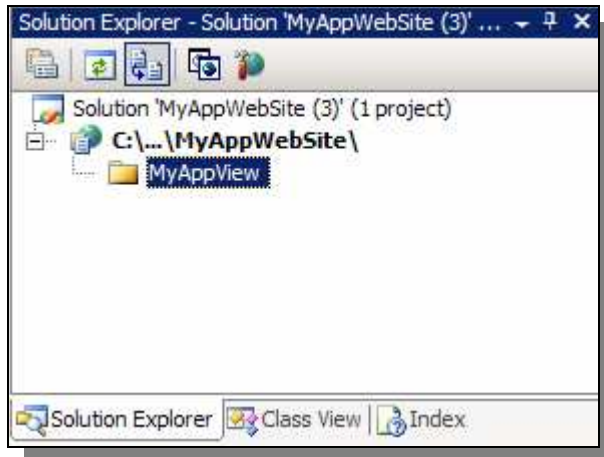
2. The site will contain two items, **Default.aspx** page and **App_Data** folder, as shown below:



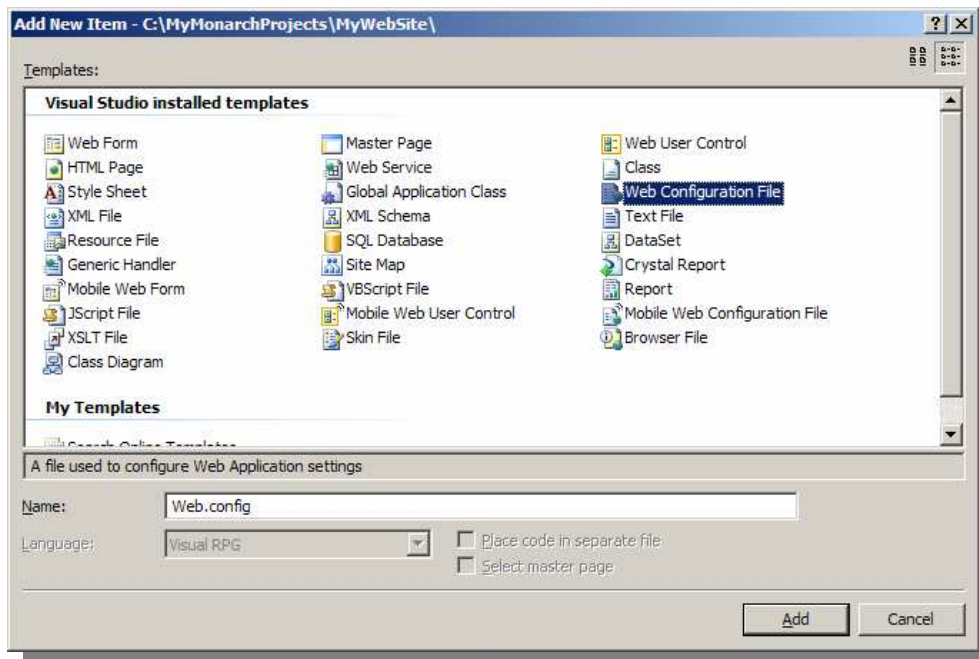
4 Creating Monarch Applications From Scratch

3. Remove the **Default.aspx** page and **App_Data** folder. Next, right-click on MyAppWebSite, then select **New Folder** from the context menu and name the new folder, **MyAppView**, as shown below.

This folder will house the future .aspx display files for the web site.



4. Add a web configuration file by selecting the **Add New Item ...** option on the **MyAppWebSite** node, and name it **Web.config**.

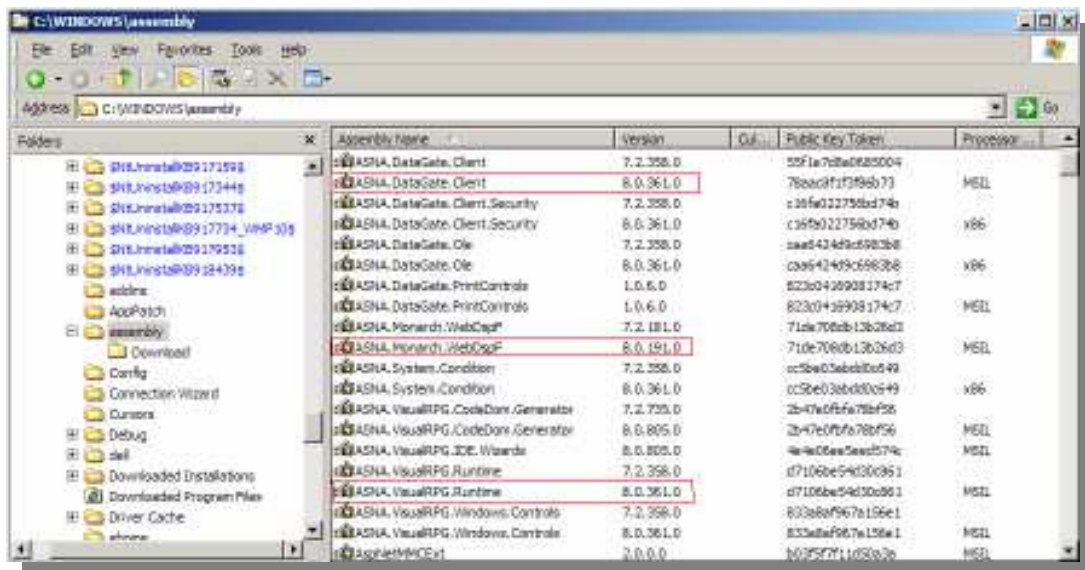


5. Open the **Web.config** file and find the `<compilation debug="false" />` element and replace it with the following code. (It is easiest to simply copy and paste from the code below). Do not forget to change `debug="false"` to `debug="true"`.

```
<compilation debug="true">
  <assemblies>
    <add assembly="ASNA.Monarch.WebDspF, Version=8.0.188.0,
      Culture=neutral, PublicKeyToken=71de708db13b26d3"/>
    <add assembly="ASNA.DataGate.Client, Version=8.0.360.0,
      Culture=neutral, PublicKeyToken=78AAC8F1F3F86B73"/>
    <add assembly="ASNA.VisualRPG.Runtime, Version=8.0.360.0,
      Culture=neutral, PublicKeyToken=D7106BE54D30C861"/>
  </assemblies>
</compilation>
```

6. Note that the assembly versions above (i.e., `Version=8.0.188.0`; `Version=8.0.360.0`) may not be the assembly versions currently installed on your machine. However, there is no harm in using lower version numbers than those installed on your machine because of backward compatibility.

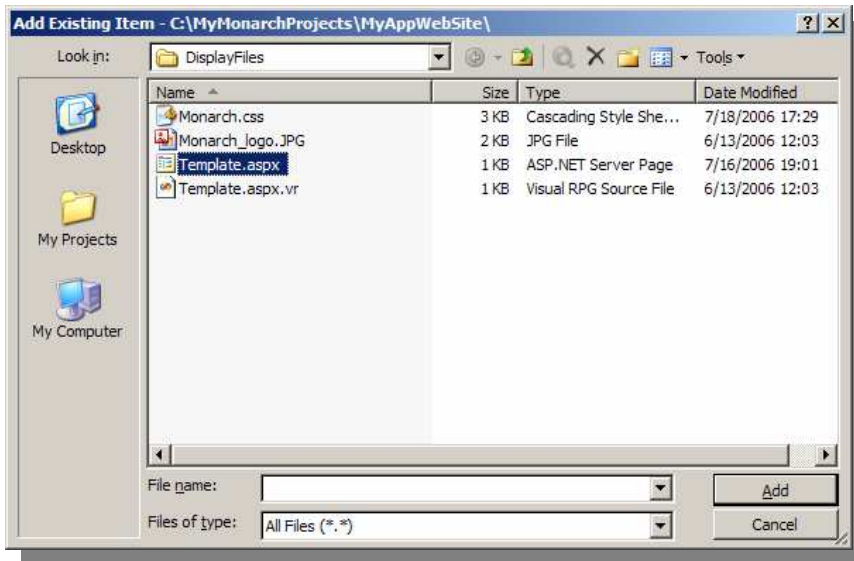
If you wish to use the currently-installed version numbers in Web.Config, you can easily locate the installed versions in the General Assembly Cache (GAC) from the Windows file explorer path, "C:\WINDOWS\assembly" as shown below:



Adding Display Files

- Right-click on **MyAppView** folder and select the **Add Existing Item...** option from the Context menu.

In the **Add Existing Item** dialog, navigate to the location you have directed Cocoon to use the **Template.aspx** page. By default, the template for display files is located in "...\Program Files\ASNA\Monarch 4.0\Cocoon\Templates\DisplayFiles" as shown below:



- After adding the file, rename it to what you would like your display file to be called. In our case, we'll call it **DspTree.aspx**.

Note: Renaming the *DspTree.aspx* file will also rename the *DspTree.aspx.vr* file.

Fixing the Code-Behind File

- Open the code-behind file by clicking the '+' sign next to **DspTree.aspx**, then double-clicking *DspTree.aspx.vr*.
- You will have to provide values to two place holders in the template that give the class its full name.

First, provide a namespace by replacing the **{0}** with your own name in the first source line. In this exercise, we'll use **MyCompany.MyTechnology.Cats.View**, as shown below.

```
DclNameSpace MyCompany.MyTechnology.Cats.View
```

- Follow that with a name for the class instead of the **{0}** in the **BeqClass** line. We will call it **DspTreeForm**. Finally, provide the parameters for the

ImpDspFile command, which provides a name to the display file and a path to the .aspx file.

```
BegClass DspTreeForm Partial( *Yes ) Extends( ASNA.Monarch.WebDspF.Page )
    ImpDspFile Name( DspTree ) DspFile( "../DspTree.aspx" )
EndClass
```

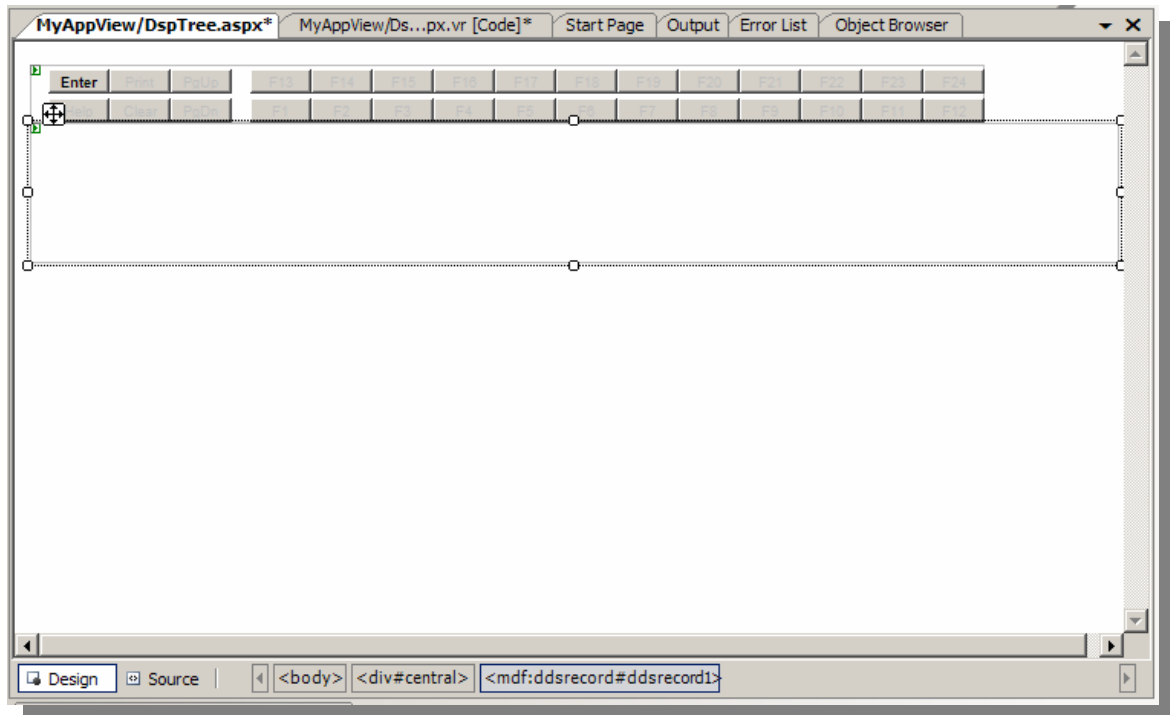
Preparing the ASPX file

12. Next, we need to initialize the **.aspx** file. Click the "Source" tab for DspTree.aspx. At the top of the file there is a **<%@ Page** directive. This line should point to the code-behind file and class. Set the **CodeFile** and the **Inherits** attributes to refer to your display file. For this exercise, the code is shown below.

Note: The value for the **Inherits** attribute is *case sensitive*.

```
<%@ Page Language="AVR" AutoEventWireup="true"
CodeFile="DspTree.aspx.vr"
Inherits="MyCompany.MyTechnology.Cats.View.DspTreeForm" %>
```

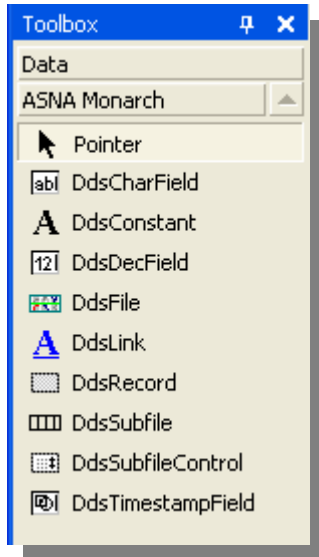
What we have now is an empty display file that looks like this in the design view.



The file has a single record format, which from the default Monarch template is called **DdsRecord1**, and has no constants or fields. When a record format is selected (as shown above), you can change its size with the mouse.

Let's add a couple of fields and constants to the format to make the exercise more interesting. You will want to make the format taller in order to add the controls listed below. See the completed web form on the next page.

ASNA Monarch Web Controls are installed onto the Toolbox as shown below. (If Monarch controls are not present on the Toolbox, you can add them manually. See [Adding Monarch Controls to VS 2005 Toolbox in the Appendix.](#))

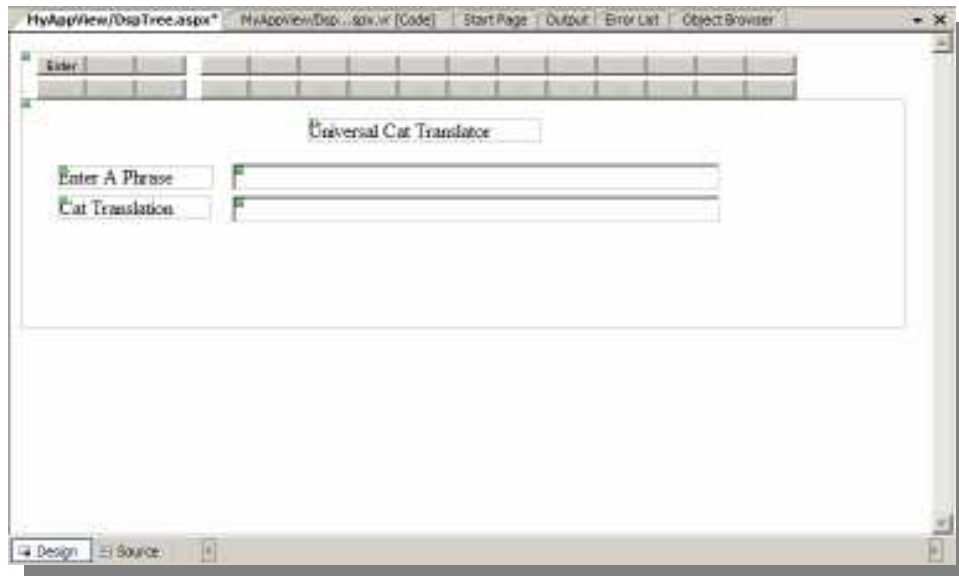


13. Drag the **DdsConstant** control onto the form to create the following constants and set their properties as listed below:

Control Name	Text
DdsConstant1	Universal Cat Translator
DdsConstant2	Enter A Phrase
DdsConstant3	Cat Translation

14. Now, drag the following two **DdsCharFields**.

Control Name	Alias	Length	Color	Protect	VirtualRowCol
DdsCharField1	Phrase	50	Red:31		0325
DdsCharField2	CatTrans	50		71	0425

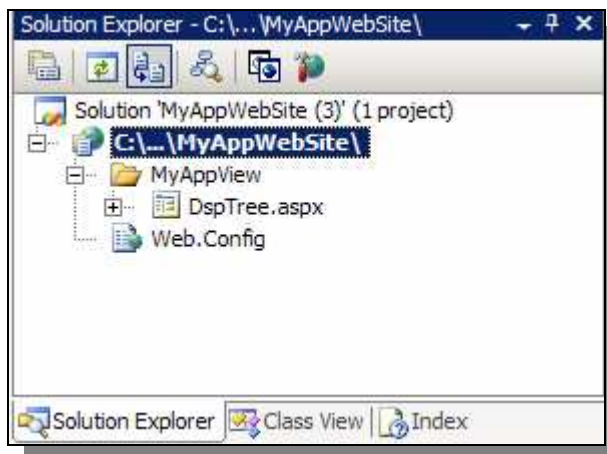


Completed Web Form

Completing the Web Site

So far, the web site contains:

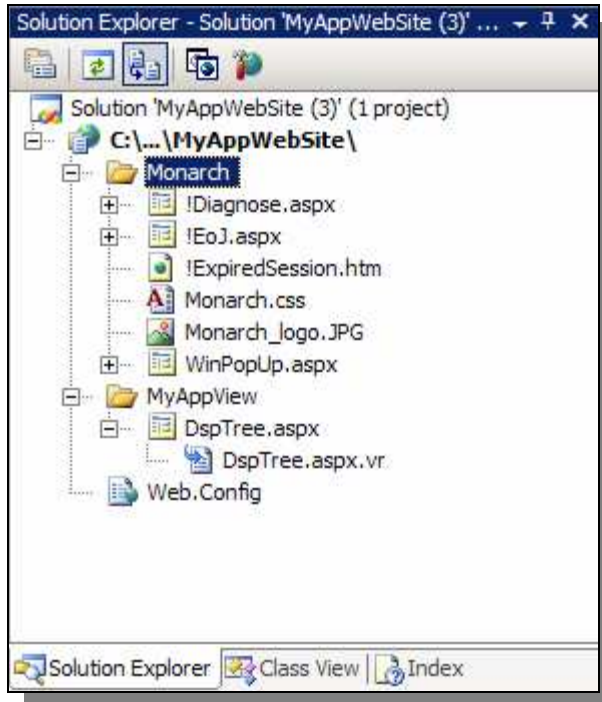
- A folder (MyAppView) with the ASPX file and its corresponding code-behind class.
- A **Web.config** file.



15. Now, it's necessary to add the set of supporting files used by Monarch to handle the special cases of end-of-job, runtime exceptions and expired sessions. There is also a page used by the record format displayed within a pop-up window. Finally, the default display file template refers to a cascading style sheet called **Monarch.css**.

All of these files reside within a folder called **Monarch** in the root directory of the web site; in our case **MyAppWebSite**. The easiest way of getting all of these files is to copy them from an existing migrated application.

*Note: If it is necessary to assemble them by hand, create a folder called **Monarch** (as shown below), then you can get them from the installation folder of Monarch's Cocoon.*

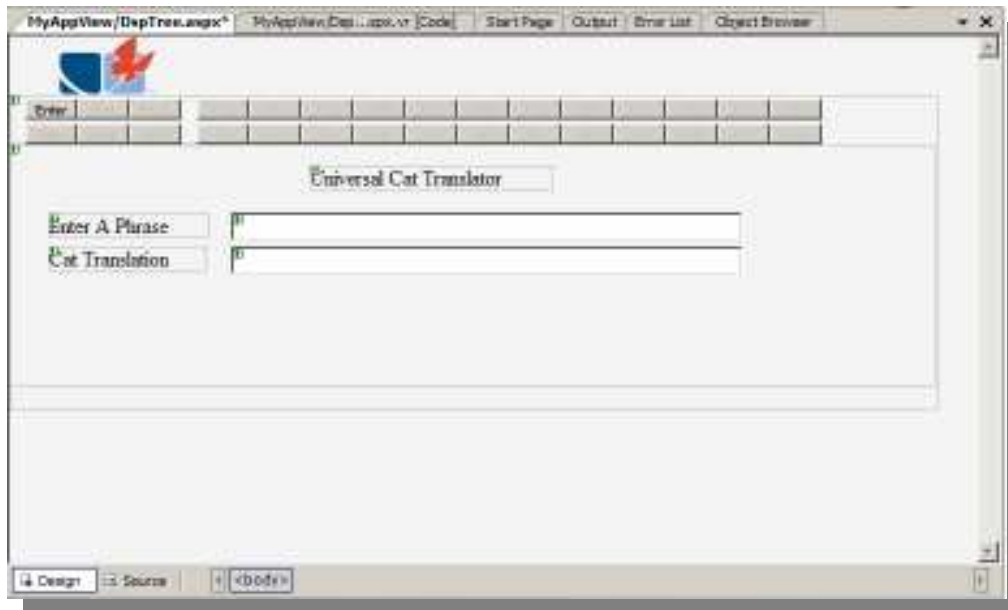


The Monarch supporting files, their function and templates location are listed below.

File	Function	Located in :
		C:\Program Files\ASNA\Monarch 4.0\Cocoon\Templates\...
!Diagnose.aspx	Target of application runtime error	AspWebApp
!EoJ.aspx	Target of application reaching end of job	AspWebApp
!ExpiredSession.htm	Target of expired sessions	AspWebApp
WinPopUp.aspx	Helper page to implement pop-up windows	AspWebApp
Monarch.css	Default cascading style sheet	DisplayFiles
Monarch_logo.jpg	Used by Monarch.css as banner title	DisplayFiles

Note: When adding an .aspx file, the corresponding .aspx.vr file is automatically added to your project.

16. After adding **Monarch.css**, switch from **Design** view for DspTree.aspx to **Source view** and then back again to **Design** view. You will notice the appearance of the page now reflects the stylesheet, like this:

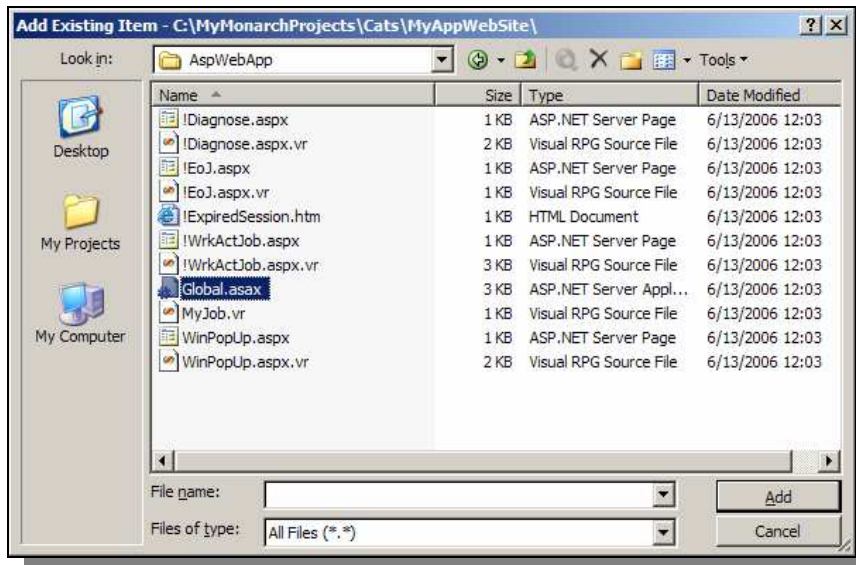


Adding Global.aspx

17. There is one more file we must add to complete our web site as the presentation layer for our application: **Global.aspx**.

In the solution explorer:

- Right-click on MyAppWebSite (C:\MyMonarchProjects\Cats\MyAppWebSite).
- Select **Add Existing Item...** from the Context menu.
- Go to: "C:\Program Files\ASNA\Monarch 4.0\Cocoon\Templates\AspWebApp".
- Select **Global.aspx**.



The main function `Global.aspx` will do for our application is to invoke the entry point of our business logic. However, we have not created the business logic class library yet, so we will leave this step incomplete for now and come back to it later in the next chapter.

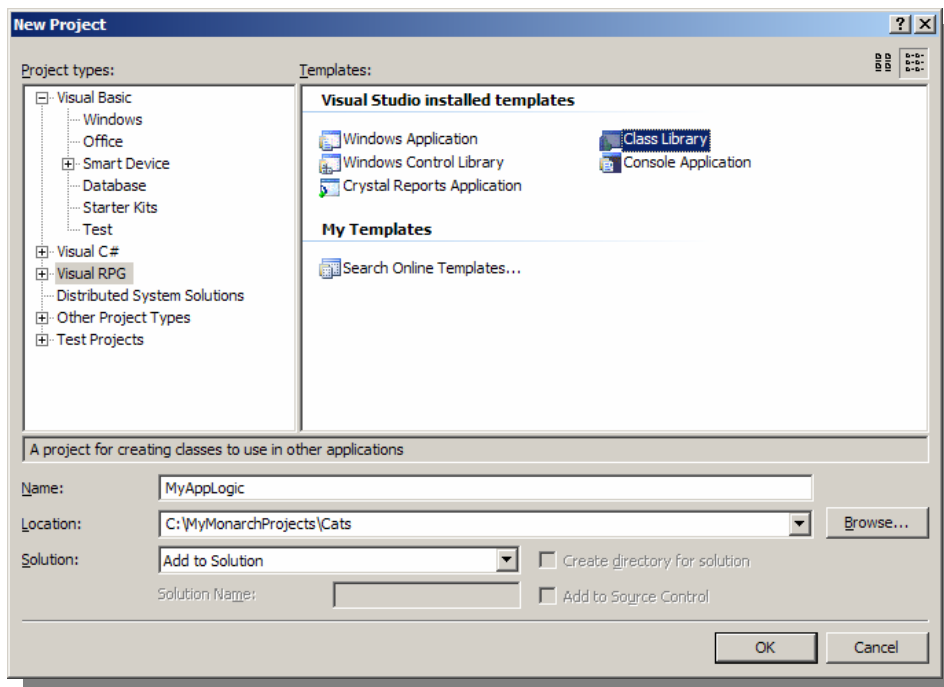
Chapter 3 – Creating the Business Logic Layer

Creating a Class Library Project

All of the programs and supporting classes needed to implement the business logic will be packaged in a Class Library.

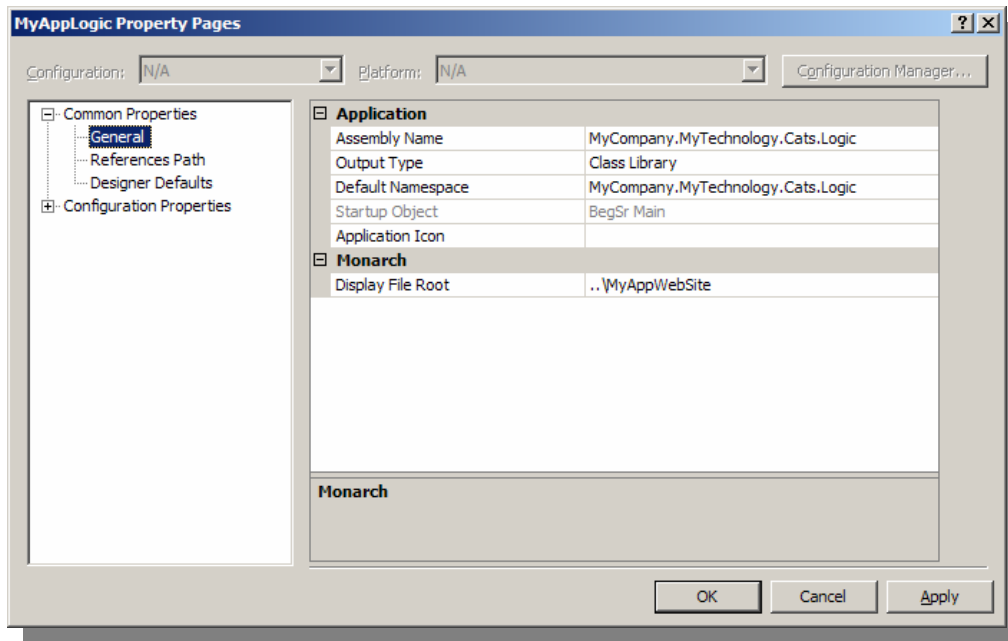
1. Create a class library by selecting the **New Project** menu option from the **File** menu (File - New - Project). When the dialog comes up, be sure to select the **Add to Solution** option and locate the Class Library as a sibling of the web site holding the presentation layer.

The following figure shows the creation of the **MyAppLogic** library under the "C:\MyMonarchProjects\Cats" folder.



2. The first thing we will do is set up some project settings by selecting the **Properties** menu option from the context menu on MyAppLogic.
 - Set **Default Namespace** and **Assembly Name** to: `'MyCompany.MyTechnology.Cats.Logic'`.
 - Set the **Display File Root** to `'..\MyAppWebSite'`.² This root is used to locate the .ASPX file used in the **DclWorkstnFile** commands of interactive programs and must point to the root web site, as shown in the following display.

² The next time you view this dialog, Display File Root will be shown as the fully resolved path. E.g.: C:\MyMonarchProjects\Cats\MyAppLogic\MyAppWebSite



Adding a Program to the Library

When the class library gets created, a single **Class1.vr** file is added containing a minimal class with the following code:

```
Using System
Using System.Text

DclNamespace MyAppLogic

BegClass Class1 Access (*Public)

EndClass
```

3. In order to convert **Class1** to a program, we must do several things.
 - ✓ First, let's rename the file to something meaningful; let's call it **Translate.vr**. (In Solution Explorer, right-click on the file and select *Rename* from the context menu).
 - ✓ Next, open the source file. Since we have given a namespace to the whole project we can remove the **DclNameSpace** command.
 - ✓ Change BegClass so that the class name is **Translate**. Since this class will be treated as a Monarch program, it must extend the **ASNA.Monarch.Program** class.
 - ✓ We probably won't be using the **Text** namespace, but will certainly be using the **ASNA.Monarch** namespace.

- ✓ Translate is an interactive program that uses **DspTree** as its display file, so we'll add a **DclWrkStnFile** command that refers to it.

This is what the source should look like now:

```
Using System
Using ASNA.Monarch

BegClass Translate Extends( ASNA.Monarch.Program ) Access(*Public)

    dclWorkStnFile DspTree DspFile( "~\MyAppView\DspTree.aspx" )

EndClass
```

Making *Translate* Callable

4. To allow the **Translate** class to be the target of the **CALL** command, it must provide a shared function called ***ENTRY**. This **Entry* provides the parameter list to the outside world and is responsible for finding or creating an activation or instance of the Translate class, then passes control to the mainline code.

The constructor of the class is typically responsible for initializing tables, reading data areas and opening files that are 'implicitly opened'. The reason for not using the real AVR capability of implicitly opening files is that Monarch interactive applications run in a multithreaded environment and is necessary to get the database connection specific to the thread running in the application. AVR's runtime is not aware of this framework, so the constructor is the place to do the 'implicit opens'.

The convention within Monarch is to create a private subroutine called **Process_Mainline_Code** (*although it could be called anything else*), which takes the place of the old mainline C specs and is invoked from **Entry*.

The ***Entry** routine is pretty much boiler-plate with the exception of the handling of the parameters specific to the program. In the case of Translate, let's say it takes two parameters, a Zoned (7,2) and a Char (50). **Entry* will:

- ✓ Locate or create an instance of Translate.
- ✓ Push an invocation on the current job.
- ✓ Copy the parameters to the Global fields of the instance.
- ✓ Invoke mainline code.
- ✓ Copy the Global fields back to the parameters.
- ✓ Pop the invocation.
- ✓ Possible dispose of the program's instance.

Below is *Entry (which you can copy and paste into your Translate class).

```

...
DclFld Count          *Zoned Len(7,2)          // unused in this example
DclFld StartPhrase   *Char   Len(50)

BegFunc *Entry Shared(*Yes) Type(*Ind) Access(*Public)
  DclSrParm Count Like(Count) By(*Reference)
  DclSrparm StartPhrase Like(StartPhrase) By(*Reference)

  DclFld programInstance Type(Translate)
  Try
    programInstance=Job.CurrentJob.FindIdleProgram (*TypeOf(Translate)) *As Translate
    Move (programInstance = *Nothing) NewActivation Type(*Boolean)
    If( NewActivation )
      programInstance = *New Translate()
    EndIf
    Job.CurrentJob.PushInvocation(programInstance)
    programInstance.Count = Count
    programInstance.StartPhrase = StartPhrase

    programInstance.Process_Mainline_Code()

  Catch Name(e) Type(ASNA.VisualRPG.Runtime.Return)

  Catch Name(ae) Type(Threading.ThreadAbortException)
    programInstance = *Nothing

  Finally
    If( programInstance <> *Nothing )
      Count = programInstance.Count
      StartPhrase = programInstance.StartPhrase
      Job.CurrentJob.PopInvocation()
      Move programInstance.StarIndicatorLR programLastRecord Type(*Ind)
      If( programLastRecord )
        programInstance.Dispose(*true)
      EndIf
    EndIf
  EndTry
  LeaveSR programLastRecord
EndFunc

```

- *Entry refers to a property of Translate class called **StarIndicatorLR**, with its code below. Add this code to the bottom of the Translate class.

```

BegProp StarIndicatorLR Type(*Ind)
  BegGet
    LeaveSr *InLR
  EndGet
EndProp

```

Doing the Real Thing

- Now we are ready to add the real business logic. This application is going to translate a phrase from any language in the world into cat language. Here is the code:

```

BegSr Process_Mainline_Code Access(*Private)
  Phrase = StartPhrase // Starting phrase
  ExFmt DdsRecord1
  DoWhile DspTree.FeedbackAID <> H'33' // 5250 Attn ID for "F3" key
  Clear Phrase
  CatTrans = "Meow miau miaw" // Genius translation
  ExFmt DdsRecord1
  EndDo
EndSr

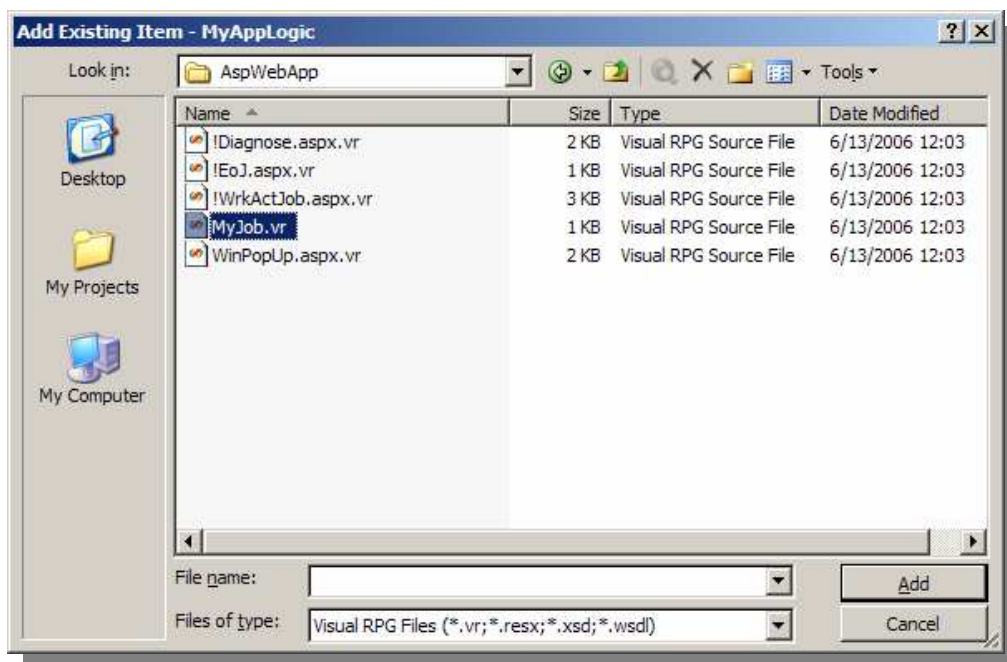
```

- This is a good time to compile the Translate class library to find any errors. Select the **Build** context menu option from the **MyAppLogic** node in Solution Explorer.
Important: This step builds **only** the MyAppLogic project, which is all that should be compiled at this time.
- Fix any syntax errors.

Getting the Ball Rolling

Now we need a job class to get the ball rolling. For that, we need to add a class that extends the **ASNA.Monarch.Job** class; which is typically called **MyJob**. Once again, the easiest thing to do is to get the class from the **AspWebApp** template folder that is installed with Monarch Cocoon.

- Select the **Add Existing Item...** context menu option on the MyAppLogic node on the Solution Explorer and browse to the templates folder.



C:\Program Files\ASNA\Monarch 4.0\Cocoon\Templates\AspWebApp

9. The **MyJob.vr** template contains several place holders that you will have to replace to customize to your situation. These are:

Holder	Function
~NewDatabase	Database name holding database files, data areas and remote programs.
~FilesWithCOMMIT_Start	Statement to start commitment control.
~EntryClassName	Class name of first program to call. <i>After the program call, don't forget to also add the DclParm statements required by your program.</i>
~FilesWithCOMMIT_End	Statement to end commitment control.

There is also a group that deals with a database connection to access print files. If your application maintains the print files in the same database as your database files, you can use the same connection as ~NewDatabase. Otherwise, you can have a separate connection for those files.

If your application does not print, you can safely remove the following place holders.

Holder	Function
~DeclareMyPrinterDB	Declare statement for MyPrinterDB, which holds the print files.
~DeclareFuncgetPrinterDB	Property Get to obtain the connection to MyPrinterDB.
~ConnectToMyPrinterDB	Statement to connect MyPrinterDB.
~DisconnectFromMyPrinterDB	Statement to disconnect MyPrinterDB.

10. Since MyAppLogic does not use commitment control and does not print, all but two of the place holders can be removed. The remaining database placeholders then are: **~NewDatabase** and **~EntryClassName**.
11. As shown in the completed code below, ~EntryClassName is replaced with the call to Translate class.

```

Using System

BegClass MyJob Extends (ASNA.Monarch.WebJob) Access(*Public)
  DclDB Name(MyDatabase) DBName("chy_5080") Access(*Public)

  BegFunc getDatabase Type(ASNA.VisualRPG.Runtime.Database)
Access(*Protected) Modifier(*Overrides)
  LeaveSR MyDatabase
EndFunc

  BegSr Dispose Access(*Public) Modifier(*Overrides)
  DclSrParm disposing Type(*Boolean)
  If disposing
    Disconnect MyDatabase

  EndIf
  *Base.Dispose(Disposing)
EndSr

  BegSr ExecuteStartupProgram Access(*Protected) Modifier(*Overrides)
  DclFld Number *Zoned Len(7,2) // For illustration only.
  DclFld Character *Char Len(50)

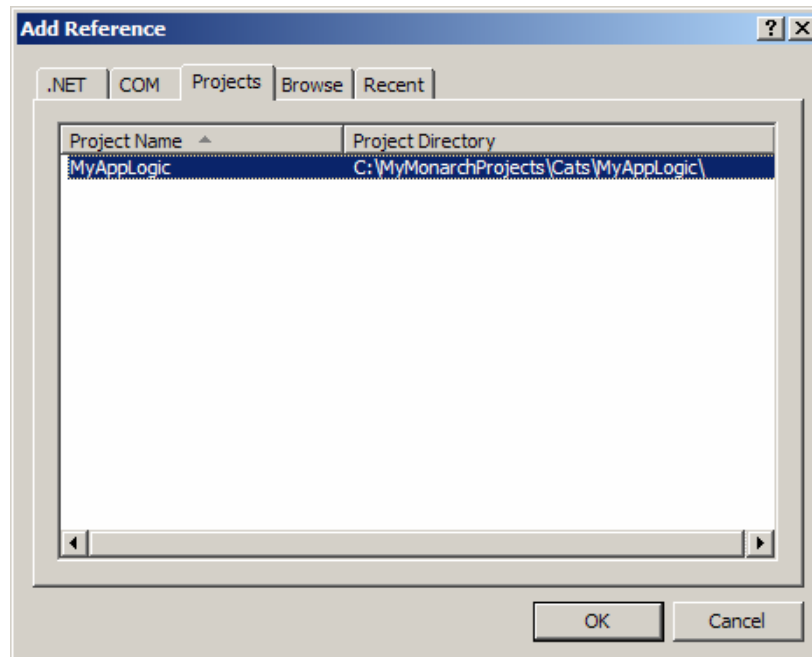
  Number = 3 // We didn't do anything with this.
  Character = "What's New Pussy Cat?"

  Connect MyDatabase
  Call Translate
  DclParm Number
  DclParm Character
EndSr
EndClass

```

Connecting Back to the Web Site

- Back to the Web site project, add **MyAppLogic** as a **Project Reference** to the site. You can right-click on **MyAppWebSite** and select **Add Reference** from the context menu.



13. In the source for Global.asax, locate the **Session_Start** subroutine and go to the line where the Job variable is being assigned a new **MyJob**. You will need to qualify the MyJob() with the namespace of MyAppLogic which is³: **MyCompany.MyTechnology.Cats.Logic**. See below:

```
Session("MonarchInitiated") = *Nothing
Job = *New MyCompany.MyTechnology.Cats.Logic.MyJob()
Session("Job") = Job
```

14. Finally, we must set one of the ASPX pages as the startup page. Set DspTree.aspx as the first page by right-clicking on DspTree.aspx and select **Set As Startup Page** in the context menu.
15. Select **Build - Rebuild Solution** to compile the entire solution.
16. Now you are ready to run the application.

³ Forgot the Namespace for MyAppLogic? Just look at the properties of MyAppLogic project for the Default Namespace.

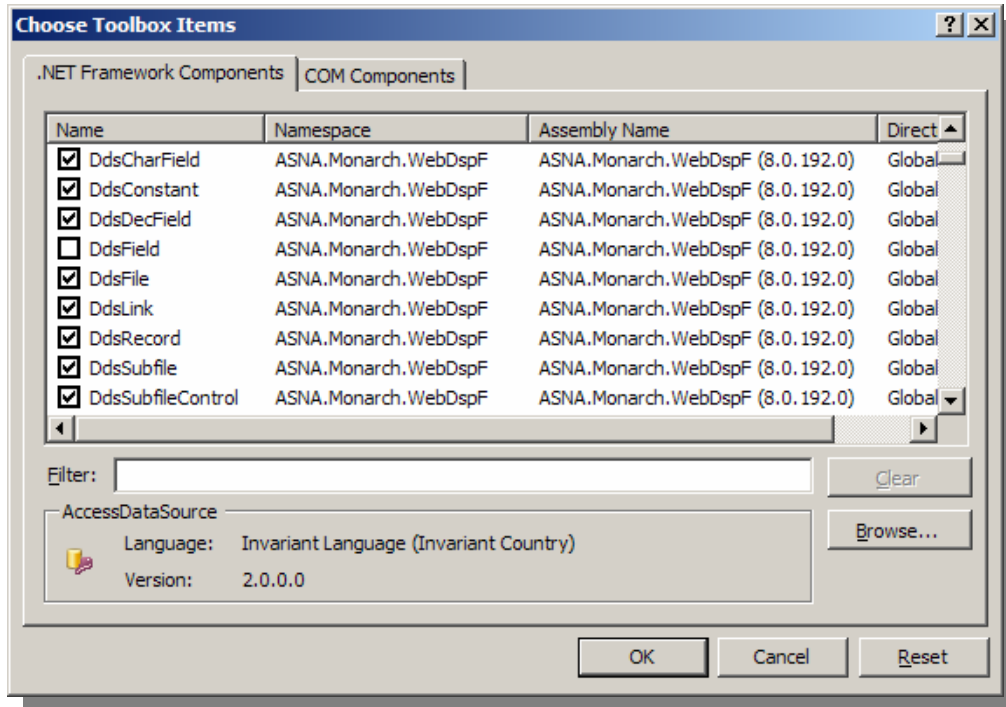
Appendix

Adding Monarch Controls to VS 2005 Toolbox

If ASNA Monarch Web Controls are not viewable from the VS 2005 Toolbox when in Web Design View, then you can add them to any Toolbox tab.

1. Select **"Choose Items..."** from the Toolbox context menu, and select the Dds controls in the ASNA.Monarch.WebDspF Namespace, as shown below.

Note: *DdsField is not included in the Toolbox.*



Hint: *Sort components by Namespace or Assembly Name to make it easier to locate ASNA.Monarch controls.*

2. Now you will see the Monarch Web Controls in the Toolbox.