Module19: Dyalog.Asp.Net

APL and its applications may yet lead the World through cyberspace! The empowering key is that Dyalog APL programs may be run remotely from practically any Internet browser on the Planet. There are no excuses now. If APL is as good as we think it is then it should begin to shine through IE7.

§ 19.1 Dynamic Web Pages

§§ 19.1.1 Active Server Pages in VBScript or JScript

The early Internet worked by a browser requesting a page from a server and the server delivering the requested page to the browser in HTML format, where a server is identified by its IP address (or corresponding domain name via a DNS) and port (generally port 80).

In 1995 Sun and Netscape incorporated Java into Netscape Navigator. Essentially, Sun added some new tags into HTML that will run Java programs inside a web browser that is *Java-enabled*. Java programs that run in web browsers are called *applets*. They heralded *dynamic* Internet sites by incorporating **client-side programs**. When you use a Java-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser's Java Virtual Machine. Employing local processing like this can enhance an Internet site in many ways, from generating drawings, graphics and animations to getting better control of the mouse, keyboard and available fonts. However, all this requires local facilities that are not yet enabled on the average browser.

Active Server Pages (ASP) introduced by Microsoft in 1996 added the complementary facility. ASP heralds *dynamic* Internet sites by incorporating **server-side programs**. When a browser requests an ASP, the web server generates, via some COM-enabled language, a page in HTML code, and sends it back to the browser. Dyalog APL can create OLEServer objects and can therefore gain access to this technology. Note that it does not rely on any special (atypical) local facilities on the client side.

19.1.1.1 Type http://82.111.24.53:8081 into your web browser or, eqivalently, select [Webserver] in www.dyalog.com to view the Dyalog.Asp example outlined below. The Dyalog web site on IP address 82.111.24.53, port 8081, if accessible from your (possibly prohibitively well protected) web location, demonstrates the Dyalog APL ASP server, called ASPSVR.

If not accessible from the remote Dyalog web site, the Dyalog APL ASPSVR may be installed and run locally on your computer.

19.1.1.2 Download ASPSVR.ZIP from the [Download Zone][Document Download Zone][ASPSVR] section of www.dyalog.com and follow the instructions for a good introduction to "classic" ASP with Dyalog APL version 9. A summary follows below.

The default HTML web page on this site is called **default.htm**. The web page involves frames and therefore it automatically requests a couple more sub-pages including **toc.htm** as highlighted below.

default.htm

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>…
<frameset cols="30%,70%">
..<frame name="toc" marginwidth="1" marginheight="1" src="toc.htm" target="main" />…
</html>
```

The DOCTYPE line is primarily there so that web authors can validate their HTML documents. There are many variations. None is necessary, but one recommended choice is for HTML 4.01 validation:
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 TRANSITIONAL//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">

The table of contents file, toc.htm, contains a link to an ASP file called **dyalog.asp**, as does another sub file in the web site, **loan.htm**. These two files call dyalog.asp in different ways. Both approaches ultimately lead to the main goal; execution of a program in Dyalog APL.

toc.htm

```
..<td><a href="loan.htm"
..<td><a href="dyalog.asp?DWSAction=driller.RUN" target="contents">Sage Driller</a></td>
..<td><a href="dyalog.asp?DWSAction=rain.Fourier&format=PNG"
target="contents">Fourier...
```

loan.htm

```
..<form action="dyalog.asp" method="GET" name="LoanForm"
..<td><input type="text" size="9" name="LoanAmt"></td>
..<td><input type="text" size="6" maxlength="6" name="PercentDown"></td>
..<p><input type="submit" value="Calculate Repayments"></p>
..<input type="hidden" name="DWSAction" value="loan.RUN">...
```

Dyalog.asp is an *active server page* file written in VBScript. The script creates an instance of an OLEServer called **dyalog.ASPSVR** that owns an exported method called **MakeHTML**. This function is passed an argument consisting of some string following a query (**?**), *eg* **DWSAction=rain.Fourier&format=PNG** in toc.htm above. **MakeHTML** runs in Dyalog APL and may execute any APL code as long as its final result is HTML. However, VBScript is generally only supported by the Microsoft browser IE (and the alternative JScript is not enabled by default).

dyalog.asp

```
<% @Language = "VBScript" %>
.. Set aplsvr = Server.CreateObject("dyalog.ASPSVR")
.. strQuery = Request.QueryString
..Response.Write aplsvr.MakeHTML (strQuery)...
```

The Dyalog OLEServer **dyalog.ASPSVR** is created from workspace **aspsvr.dws**. [File][Export] generates **aspsvr.dll**, which has to be registered on the server using **regsvr32.exe**. When used, aspsvr.dll is run in conjunction with **C:\WINDOWS\system32\dyalog.dll**. Within the aspsvr.dws workspace, the function *#.ASPSVR.MakeHTML* is exported as a method. The essence of this function is:

```
∇HTML←MakeHTML PARS;NS;FN;DWSPARS;INST;⎕IO;⎕ML
.. DWSPARS PARS←SplitPars DeCode⍣¯1'&='Split PARS
.. HTML←⍎FN,' ↑PARS'
...∇
```

Once registered, the OLEServer may be tested in Dyalog.

```
      'ASV'⎕WC'OLEServer' 'dyalog.ASPSVR'
      ⍴ASV.MakeHTML'DWSAction=loan.RUN'↳3759
      ⍴ASV.MakePicture'DWSAction=rain.Timeseries&format=PNG'↳204
```

In the Dyalog.asp example above, this OLEServer is used to drive a web server that can execute Dyalog APL code remotely. This is reminicent of mainframe timesharing for 1st generation APLs ☺.

§§ 19.1.2 The `System.Web.UI.Page` Class

In 2002 "classic" ASP was superceded by ASP.NET. ASP.NET is part of Microsoft .NET and therefore has full .NET support for languages such as VisualBasic.NET, C#, .. and Dyalog APL.

In Windows, an internet site is hosted by IIS, the Internet Information Service. IIS must be installed from the Windows CD *before* .NET is installed. And .NET must be intalled *before* Dyalog APL and the Dyalog APL 11.0 .Net Interface Components. Note that at least in beta versions of Dyalog APL 11.0, IIS 6.0 in combination with .NET 2.0 is not supported, and ASP.NET may only be used with Dyalog APL when using .NET 1.1 plus SP1, as matches the default version specified in **dyalog.exe.config**.

An ASP.NET page is identified by the extension **.aspx** (as opposed to **.asp** for classic ASP). All executable code is moved out of the <html> section and into a new <script> section.

As far as the programmer is concerned, the dynamic part of ASP.NET pages is built with graphical controls in a way similar to a standard Windows user interface, and the program dynamics is event-driven like all Windows GUI applications. For example, a web button is assigned properties and responds to events. (However, rather than being immediately drawn, web controls in segments of *html plus script* are built on the server and form part of the resulting page sent to the end-user's browser.)

On receipt of a request for a .aspx page, the ASP.NET engine within IIS automatically creates a class that derives from the `System.Web.UI.Page` class. The dynamically created class is immediately compiled, and ultimately produces html which is returned to the client.

Multi-user access is managed by IIS. In particular, IIS maintains one distinct AppDomain for each ASP.NET application currently running.

19.1.2.1 Copy the following code into Notepad and save the file as **C:\Inetpub\wwwroot\VB1.aspx**, then type http://localhost/vb1.aspx in IE. A button asking to be clicked (*cf* kicked or flicked) should appear.

C:\Inetpub\wwwroot\VB1.aspx

```
<html>
 <body>
  <form runat="server">
   <asp:Button id="button1" Text="Click me!" runat="server" />
  </form>
 </body>
</html>
```

In ASP.NET, all HTML server controls must be within a single <form> tag with the **runat** attribute set to "server". Note that there can only be one **<form** runat="server"> control per .aspx page. The runat="server" attribute indicates that the control should be processed on the server. It also indicates that the enclosed (in < .. >) controls may be accessed by server scripts.

19.1.2.2 Add the VB script section in pink to vb1.aspx. Note the addition of the **OnClick** event that initiates the VB submit callback when the button is pressed. Save as **C:\Inetpub\wwwroot\VB1.aspx** and press the button from http://localhost/vb1.aspx. Note that all executable code resides *outside* the <html> tags.

C:\Inetpub\wwwroot\VB1.aspx

```
<script Language="VB" runat="server">
  Sub submit(Source As Object, e As EventArgs)
  button1.Text="You clicked me!"
  End Sub
</script>
```

```
<html>
 <body>
  <form runat="server">
   <asp:Button id="button1" Text="Click me!" runat="server" OnClick="submit"/>
  </form>
 </body>
</html>
```

Within the submit function all sorts of other VB code could be added, eg

```
button1.Style("background-color")="#0000ff"
button1.Style("color")="#ffffff"
button1.Style("width")="200px"
button1.Style("cursor")="hand"
button1.Style("font-family")="verdana"
button1.Style("font-weight")="bold"
button1.Style("font-size")="14pt"
button1.Text="A New Caption"
```

A second example of the OnClick event, this time involving a TextBox control, is shown in VB2.aspx below. You will convert this example from VB, the default language, to Dyalog in exercise 19.2.1.1.

C:\Inetpub\wwwroot\VB2.aspx

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lab1.Text="Your name is " & txt1.Text
End Sub
</script>
<html>
<body>
<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lab1" runat="server" /></p>
</form>
</body>
</html>
```

When a browser makes a request for an ASP.NET web page, the request is first sent to the server implicated by the URL. If it is a Windows server, IIS receives the request, recognises the .aspx extension, and passes the request on to ASP.NET for processing. ASP.NET creates an instance of the System.Web.UI.Page class from the .aspx file contents.

When a page is created the Load event is triggered. By default, ASP.NET tries to find the special method name Page_Load on the page. If a match is found, the function is considered to be a handler for the Load event. In other words, Page_Load is taken to be the callback attached to the Load event.

19.1.2.3 Add the following file to your default web site and run it IE. Notice how the time changes on refresh.

C:\Inetpub\wwwroot\VB3.aspx

```
<script runat="server">
 Sub Page_Load
lab1.Text="The date and time is " & now()
End Sub
</script>
```

```
<html>
<body>
<form runat="server">
<h3><asp:label id="lab1" runat="server" /></h3>
</form>
</body>
</html>
```

The Page_Load subroutine runs every time the page is loaded. If you want to execute the code in the Page_Load subroutine only the first time the page is loaded, you can use the **IsPostBack** property of a Page object – *ie* an instance of the Page class. If the Page.IsPostBack property is false (0), then the page is being loaded for the first time. If IsPostBack is true (1), the page is being posted back again to the server.

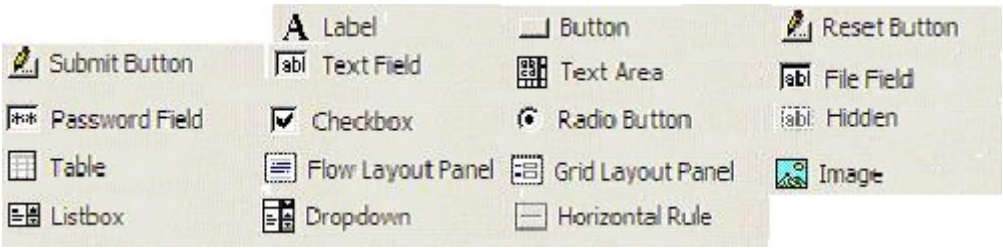
19.1.2.4 Incorporate the 'If' condition into **VB3.aspx** and notice the different response on [View][Refresh].

```
<script runat="server">
Sub Page_Load
If Not Page.IsPostBack then
  lab1.Text="The date and time is " & now()
End If
End Sub
```

§§ 19.1.3 The *System.Web.UI.WebControls* Namespace

There are two groups of controls available to the web programmer. There are the standard ones used to present web pages, including dynamic ones for client-side scripts. And there is the new set of dynamic controls for ASP.NET server-side scripts.

Under .NET, the first group is located in the **System.Web.UI.HtmlControls** namespace. These map directly to standard HTML tags supported by all browsers. They allow simple programmatic control of HTML elements on any HTML or ASP.NET page. The second group is ASP.NET specific and is found in the **System.Web.UI.WebControls** namespace. Here is a list of some of them. They are distinguished by the fact that they may be used to initiate a program *on the server*.



These web controls may be included in the .aspx <form> tag like this:

```
<asp:HyperLink id="HyperLink1" runat="server">HyperLink</asp:HyperLink>
<asp:RadioButtonList id="RadioButtonList1" runat="server"></asp:RadioButtonList>
<asp:DropDownList id="DropDownList1" runat="server"></asp:DropDownList>
<asp:ListBox id="ListBox1" runat="server"></asp:ListBox>
<asp:Image id="Image1" runat="server"></asp:Image>
<asp:AdRotator id="AdRotator1" runat="server"></asp:AdRotator>
<asp:Table id="Table1" runat="server"></asp:Table>
<asp:Calendar id="Calendar1" runat="server"></asp:Calendar>
<asp:DataGrid id="DataGrid1" runat="server"></asp:DataGrid>
```

Usually callbacks are set on appropriate events on these controls. For example, a form is most often submitted by clicking on a button.

```
<asp:Button id="id" text="label" OnClick="submit" runat="server" />
```

There is also a set of controls whose job it is to validate entry into certain web controls. eg

```
<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"
                            ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator>
<asp:CustomValidator id="CustomValidator1" runat="server"
                             ErrorMessage="CustomValidator"></asp:CustomValidator>
<asp:ValidationSummary id="ValidationSummary1" runat="server"></asp:ValidationSummary>
```

For example, to restrict the contents of a TextBox called tbox1 to integers between 1 and 100, a RangeValidator may be set to

```
<asp:RangeValidator ControlToValidate="tbox1" MinimumValue="1" MaximumValue="100"
        Type="Integer" EnableClientScript="false" Text="The value must be from 1 to 100!"
        runat="server" />
```

A more complete list of controls can be found at the official ASP.NET web site, http://www.asp.net.

§ 19.2 Dyalog Script Language

§§ 19.2.1 Callbacks in Dyalog APL

The script in file **C:\Inetpub\wwwroot\VB1.aspx** is in the default scripting language, VB .NET. In order to rewrite it in Dyalog APL it is necessary to set the Language attribute to **"dyalog"** in Dyalog version 11 (or "apl" in version 10). Then all that needs to be done is to convert the Visual Basic .NET code into Dyalog APL code.

C:\Inetpub\wwwroot\APL1.aspx

```
<script language="dyalog" runat="server">
   ∇submit args
   :Access Public
   :Signature submit Object Source, EventArgs e
   button1.Text←'You clicked me!'
   ∇
</script>
<html><body>
  <form runat="server">
    <asp:Button id="button1" Text="Click me!" runat="server" OnClick="submit"/>
 </form>
</body></html>
```

The `:Access Public` statement means the function may be called from outside the script.

The `:Signature` .. statement is the equivalent of `Source As Object, e As EventArgs` and defines the types of the standard incoming event message arguments. These arguments are not actually used in this particular APL code, although the third line could have been coded as

```
(⊃args).Text←'You clicked me!'
```

19.2.1.1 Convert **C:\Inetpub\wwwroot\VB2.aspx** to **C:\Inetpub\wwwroot\APL2.aspx** and test it in IE.
Hint: See [Control Panel][Regional and Language...][Languages][Details][Settings] and the *Dyalog.Net* Manual Chapter 10 for scripting APL in Notepad.

Note that ⎕USING may be assigned inside the <script> tags, indicating that the full power of the .NET framework as well as the full power of Dyalog APL may potentially be summonsed from any browser.

§§ 19.2.2 Workspace behind …

19.2.2.1 Start the Dyalog.Net tutorial at www.dyalog.com by selecting [Products][Dyalog for Windows][Microsoft .NET Interface][Web Pages Tutorial] or by typing http://82.111.24.53/tutorial.net directly into your browser. Run the examples and view the explanation of each.

In the example ..\tutorial\intro6.aspx, the entire <script> section is replaced with a reference to a workspace, **fruit.dws**, which contains a single namespace called `FruitSelection`.

19.2.2.2 Copy the file **..\Samples\asp.net\tutorial\intro6.aspx** to **C:\Inetpub\wwwroot\intro6a.aspx** and change the name of the workspace being called to **C:\Inetpub\wwwroot\fruity.dws**

C:\Inetpub\wwwroot\intro6a.aspx

```
<%@Page Language="Dyalog"
        Inherits="FruitSelection"
        src="fruity.dws" %>
<html>
<h1>intro6: Workspace Behind</h1>
<p>This example illustrates how you can use an APL workspace.</p>
<body>
        <form runat="server" >
                <asp:DropDownList
                        id="list"
                        runat="server"
                        autopostback="true"
                        OnSelectedIndexChanged="Select"/>
                <p>
                <asp:Label
                        id=out
                        runat="server" />
                </p>
        </form>
</body>
</html>
```

The only function explicitly called from the workspace is the callback, `Select`, on the DropDownList.

19.2.2.3 Create a new workspace called **C:\Inetpub\wwwroot\fruity.dws** and within it create the following `NetType` object and functions (exported as methods), then navigate to http://localhost/intro6a.aspx .

```
      )WSID C:\Inetpub\wwwroot\fruity.dws
was CLEAR WS
      ⎕using←'' 'System.Web.UI,System.Web.dll'
      'FruitSelection'⎕WC'NetType'('BaseClass' 'Page')
      )cs FruitSelection
#.FruitSelection
```

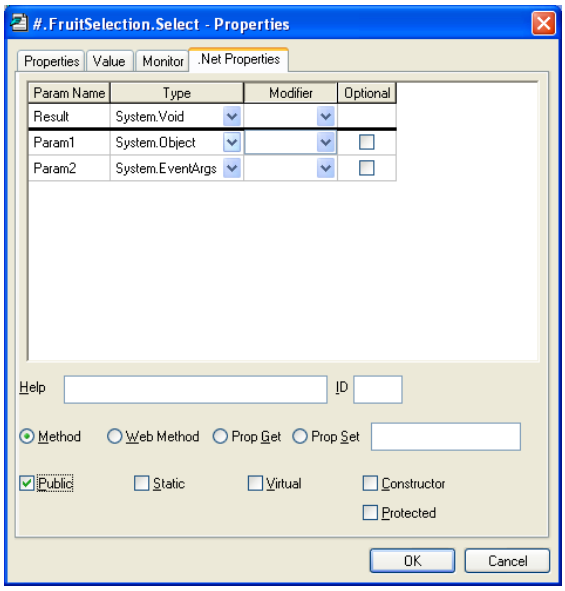
The only function explicitly called from the workspace is the callback, `Select`, which we define as:

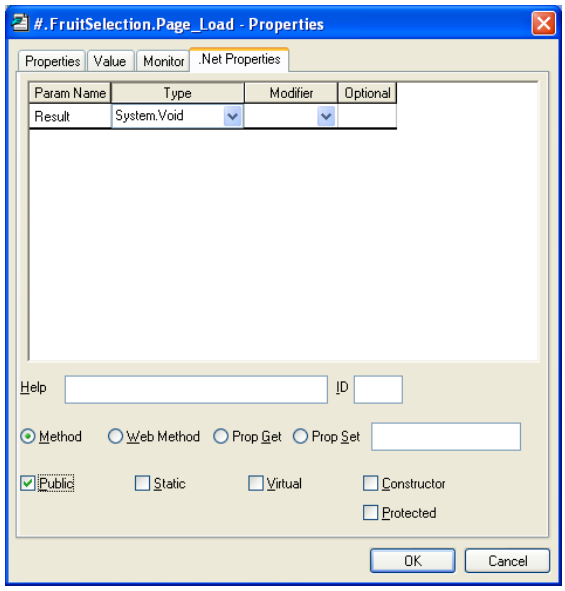
```
    ∇ Select args
      out.Text←'You selected ',list.SelectedItem.Text
    ∇
```

The contents of the DropDownList control is initially empty. We can use the `Page_Load` callback to initialise the control contents.

```
    ∇ Page_Load
      :If 0=IsPostBack
          list.Items.Add⊂'Raspberry'
          list.Items.Add⊂'Blackberry'
          list.Items.Add⊂'Grape'
          list.Items.Add⊂'Mango'
      :EndIf
    ∇
```

These two methods must be exported as Public methods and their calling structure must be set appropriately in the [Properties][.Net Properties] popup boxes (from right-clicking on (`CurObj`) function).





As shown in the online tutorial, the APL code may alternatively be saved as a class in a *script file* with extension **.apl** in place of a workspace. This is more consistent with standard language methodology, called *code behind* rather than *workspace behind*, but it loses the many advantages of APL workspaces.

(Alternatively, a *primitive APL class* called `FruitSelection` may replace the `NetType` object. This approach is more consistent with the latest standard language methodology and is followed in §20.)

19.2.2.4 Convert **..\tutorial\into1.aspx** to use a "workspace behind" rather than a scripted function.

§§ 19.2.3 The TextBox Control

We are now in a position to implement a very basic APL session hosted inside IE. The session window might be represented by a TextBox control whose TextMode attribute is set to "multiline".

```
.. <body style="font: 10pt verdana">
    <form runat="server">
     <h3>Dyalog ASCII</h3>
      ..<asp:TextBox id="txt1" textmode="multiline"
                     runat="server" rows="20" cols="50"
                    acceptsReturn="1"></asp:TextBox>…
    </form></body>…
```

A line typed into this TextBox may be executed on the server in Dyalog APL. A button may be used to initiate execution. Note that no APL font can be assumed to exist on an arbitrary local machine.

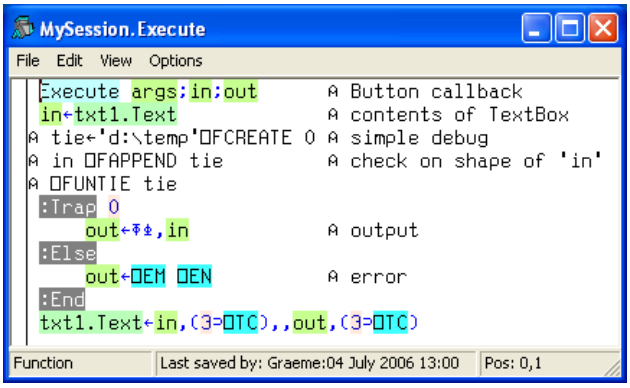
```
..<p>
  <asp:Button id="btn1" Text="Execute" runat="server"
              NotifyDefault="1" onclick="Execute"/>
  </p>...
```

The <script> section is replaced by a file with an opening line such as:

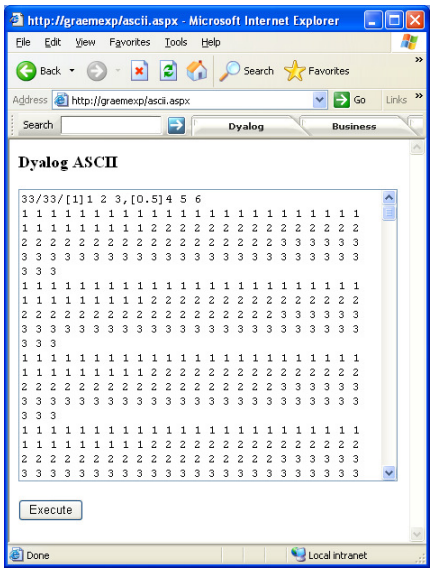
```
<%@Page Language="dyalog" Inherits="MySession" src="ascii.dws" %>
```

The workspace **ascii.dws** contains a `NetType` object called `MySession` whose `BaseClass` is the System.Web.UI.Page class. `MySession` contains just one function, `Execute` that is exported with properties similar to `#.FruitSelection.Select` above.

Inside the APL `Execute` function, the text in the TextBox is extracted from `txt1.Text`. The line text is executed and the result (or `⎕DM` in the event of an error) is inserted back into `txt1.Text`.



19.2.3.1 Write a page called **..\ascii.aspx** which invokes a "workspace behind" called **ascii.dws** that executes lines of ASCII text as if they were in some APL font.



This has the effect of allowing parentheses, brackets, quotes and some primitives such as `+ - × ! ? | , ~ < = > ∧ \ / & .` and `#` to be typed in directly.

Other APL primitives would be usable (as different symbols such as ½ for ⍴ or ¼ for ⍳ or „ for ←) if only they could be typed or pasted into a TextBox in IE. What can be typed into a typical Windows application depends on the Windows Language in operation. In XP, this is found in [Control Panel][Regional and Language …][Languages][Details][Settings]. Notice the **Dyalog APL New Keyboard** is installed here. This enables you to write APL scripts in Notepad. Note, however, that this *Input Method Editor* is only present on a computer that has Dyalog installed and is unavailable in a general web setting. The availability of different scripts in IE is constantly improving, as can be seen from Microsoft web site: http://www.microsoft.com/windows/ie/ie6/downloads/recommended/ime/ . Perhaps, in Vista, Unicode will be fully supported for input into IE in any language, but keyboard limitations are likely to be the main constraint (perhaps until they move from 8 bit to 16 bit devices). (Even applications such as Microsoft Word that store text in Unicode generally rely on 8-bit keyboards.)

Recipients of mail to the dyalogusers group on Yahoo and readers of Vector (the Journal of the British APL Association - see http://www.Vector.org.uk) will know that Stephen Taylor has developed an APL 'sandbox' along the lines outlined above and it is accessible through the Internet.

§ 19.3 Remote Applications

Our programming future might be writing APL applications on the Internet. APL can now support this.

We first saw it first using `TCPSocket` objects in §§15.2.3 in the workspace **APLSERVE.DWS**. The main hurdle there was interpretation of the Hypertext Transfer Protocol that surrounds browser packets (as in most recently defined in RFC2616 for HTTP/1.1, dated June 1999). This is what IIS handles and is therefore not necessarily a hurdle for an APL web server. However, it will have to converse with browsers, which basically talk Hypertext Markup Language (HTML, including scripting).

Then we did it through **dyalog.ASPSRV**, an OLEServer control that we called through a classic ASP page in a file called Dyalog.asp written in VB Script. But the OLEServer has to be installed and registered on each local computer, and VB Script has to be enabled on the browser.

Now we are running APL and **ASP.NET** through IIS. This assumes very little about the client browser. The ability to support *zero-footprint clients* would seem to be the main APL route to 'everywhere' ☺.

All specifically APL web site considerations have been forced onto the server side. Browsers may therefore be viewed as extensions of APL-supported hardware - like keyboards, screens or printers - and the business of writing APL programs on the Internet platform now can begin in earnest.

The primary thrust of .NET would seem to be **ASP.NET, ADO.NET** and the facilitation of **web services**. Microsoft has been aiming at the Internet since 1996, first through its abstract DNA (Distributed interNet Architecture) methodology - a way to think about writing applications – and then through ASP whereby a set of technologies implementing a DNA solution are glued together and distributed over the web.

Now the novety in .NET seems to mostly concern the Internet. `System.Web` is the primary new functionality in .NET. The first Microsoft Internet site was born in early 1993 and launched its public Internet Web domain with a home page in 1994. In 1995 Bill Gates commented, "Amazingly, it is easier to find information on the Web than it is to find information on the Microsoft Corporate Network!" And in the same year Microsoft Internet Explorer 1.0 barged into Windows. As has been said of the evolution of mankind, Bill might say "We got here as soon as we could!"

§§ 19.3.1 The C:\Inetpub\wwwroot\ Directory

Who is going to serve your Active APL Host (Aah!)? In order to follow the route outlined above, you will need an ISP that supports the .NET framework, and ASP.NET, and IIS (*ie* Windows), and runs Dyalog APL. Each requirement reduces the list of providers and increases the cost of the web site. The best solution is to **host you own** site on a dedicated box in the corner, suitably isolated, and protected.

19.3.1.1 Obtain a static IP address that can be pinged from outside world. Give your IP address an available name, such as **apl4.net**, and register the name with a Domain Name Server. (For limited use on a private Intranet you may instead use your computer's *full name* as is given in [Control Panel][System].)

19.3.1.2 Deploy your web site by copying files such as **index.aspx** into directory **C:\Inetpub\wwwroot**. Try to access your site from the outside world by typing http://apl4.net. Once you are able to run APL like this then it is time to plan *your journey to outer-cyberspace*.

In the mid '80s there was a debate within I.P.Sharp Associates (IPSA) of Canada as to which was the more important, the IPSharp Communications Network which encircled the World and carried electronic mail and data, or Sharp APL, the leading APL language of the time. Timesharing collapsed with the appearance of PCs and STSC APL*PLUS/PC. SharpAPL/PC was too slow to be useful and Reuters bought out IPSA in 1987. Not surprisingly, it transpires Reuters just wanted the comms network.

In reality, one complemented the other. A communication system without storage of state information, like the old telephone system, is of limited use. And an undistributed computer language has limited value. The early manifestation of the Internet was like the old telephone network. Information was passed around but little if any input was saved and processed. The move now is towards a ***stateful*** Internet that remotely remembers the state between messages.

If you look at [View][Source] in IE for a **.aspx** page you will find that **<asp: ..>** contols are converted to hidden standard HTML controls when the page is sent. These hidden controls store information about the state of the ASP.NET controls. For example, you might find source HTML like:

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
  <input type="hidden" name="__VIEWSTATE"
        value="dDwtNTI0ODU5MDE1Ozs+ZBCF2ryjMpeVgUrY2eTj79HNl4Q=" />…
```

_VIEWSTATE holds the detailed status of the page sent by the server. The status is defined through a hidden field placed on each page which has a **<form runat="server">** control.

Another attempt to save basic state data is by way of local *cookies*. (While developing a web page it is sometimes necessary to delete all cookies by IE [Tools][Internet Options…][Delete Cookies…] in order to get IE to respect your program changes.)

ASP.NET adds another mechanism for storing state information. There is a .NET session class called `System.Web.SessionState.HttpSessionState` ¢. It allows storage between transactions. The excellent tutorial in http://localhost/dyalog.net/tutorial/ has examples of the `Session` object.

Another important leg of Microsoft's .NET march onto the web is **ADO.NET**. This is all about storage and retrieval of larger volumes of information. But once inside APL, storage of data is not a problem. The parent namespace of the current AppDomain (##) may be used to store temporary individual user information. Most pertinently, native or APL component files are suitable for voluminous permanent records, although, of course, many other database systems can be handled through APL via **SQAPL**.

§§ 19.3.2 The *System.Drawing* Namespace

19.3.2.1 Convert the ∇scribble∇ function in §17.3.1 into a drawing on your web site.

Sierpinski's gasket is named after Polish mathematician Waclaw Sierpinski (1882-1969). His gasket, or fractal triangle, is constructed by taking an equilateral triangle, dividing it into four smaller equilateral triangles, removing the centre triangle and repeating the process with each of the smaller triangles.

A algorithmic version for creating an approximation to Sierpkinski's gasket goes something like this:

1. Create a triangle, labelling each point of the triangle as P1, P2, and P3.
2. Pick a point within the triangle - call it CurrentPoint.
3. Randomly choose a number between 1 and 3.
4. If the value is 1, move CurrentPoint to the mid-point of the line between CurrentPoint and P1.
5. If the value is 2, move CurrentPoint to the mid-point of the line between CurrentPoint and P2.
6. If the value is 3, move CurrentPoint to the mid-point of the line between CurrentPoint and P3.
7. Draw a pixel at the new CurrentPoint.
8. Return to Step 3 (more returns give a sharper the image).

This algorithm is implemented in C# code which may be run in IE to produce the picture below.

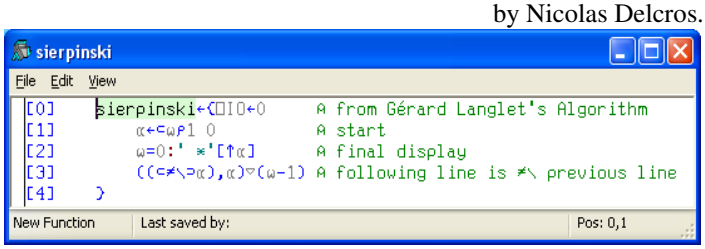
C:\Inetpub\wwwroot\serp.aspx

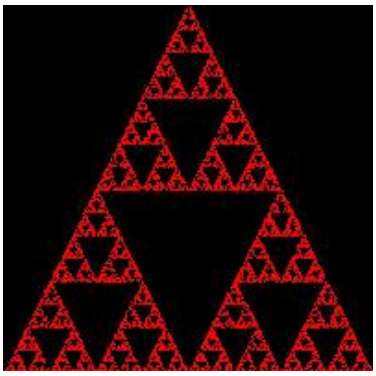
```
<%@ Page Language="c#" %>
<%@ import Namespace="System.Drawing" %>
<%@ import Namespace="System.Drawing.Imaging" %>
<script runat="server">
   void Sierpinski(int width, int height, int iterations)
   { // create the Bitmap
     Bitmap bitmap = new Bitmap(width, height);
     // Create our triangle's three Points
     Point top = new Point(width / 2, 0),
           bottomLeft = new Point(0,height),
           bottomRight = new Point(width, height);
     // Now, choose our starting point
     Point current = new Point(width / 2, height / 2);
     // Iterate iterations times
     Random rnd = new Random();
     for (int iLoop = 0; iLoop < iterations; iLoop++)
     {  // draw the pixel
       bitmap.SetPixel(current.X, current.Y, Color.Red);
       // Choose our next pixel
       switch (rnd.Next(3))
       {  case 0:
            current.X -= (current.X - top.X) / 2;
            current.Y -= (current.Y - top.Y) / 2;
            break;
          case 1:
            current.X -= (current.X - bottomLeft.X) / 2;
            current.Y -= (current.Y - bottomLeft.Y) / 2;
            break;
          case 2:
            current.X -= (current.X - bottomRight.X) / 2;
            current.Y -= (current.Y - bottomRight.Y) / 2;
            break;
       }
     }
```

```
    // Save the image to the OutputStream
    Response.ContentType = "image/jpeg";
    bitmap.Save(Response.OutputStream, ImageFormat.Jpeg);
    // clean up...
    bitmap.Dispose();
  }
  void Page_Load(Object senter, EventArgs e)
  {
    Sierpinski(200,200, 10000);
  }
</script>
<html>
<head></head>
<body></body>
</html>
```

19.3.2.2 Rewrite **serp.aspx** in Dyalog.Asp.Net.

Hint: Consider incorporating the DFn below, beautifully crafted by Nicolas Delcros.





§§ 19.3.3 The *System.Web.Services* Namespace

A **.aspx** file is a prescription for a Web Page via classes in the the **System.Web.UI** namespace. The client-server communication can then be considered at the HTML level. A Web Page is a class that expresses its functionality (properties/methods/events) through a standard web browser.

A **.asmx** file is a prescription for a Web Service via classes found in the **System.Web.Services** namespace. The client-server communication can be considered at the XML level (rather than the deeper HTTP level). A Web Service is a class that exposes its functionality (properties/methods/events) over the Internet. IE can give a basic rendering of such a service, but generally the client is expected to cater for the service through an explicit local client interface application.

For both Web Pages and Web Services, Dyalog APL code is controlled and run by the ASP.NET engine inside Microsoft IIS.

More generally, web services form the foundation of Microsoft's interoperability efforts. Apparently, Windows Vista will support XML level interaction with web-service-enabled devices, such as printers, digital cameras, and home control systems.

A .asmx file defines a class. It looks rather like a .apl file that defines a namespace in script form, except that it begins with a line looking like the opening line of a .aspx file. In the case of .asmx, this opening statement in the script file declares the language and the name of the service. For example, the following statement declares a Dyalog APL Web Service named GolfService.

```
<%@ WebService Language="Dyalog" Class="GolfService" %>
```

Details of this excellent example may be found in the *Dyalog.Net Interface* manual, chapters 6 and 7, and working code may be called from C:\Program Files\Dyalog\Dyalog APL 11.0\Samples\asp.net\golf\.

The following very simple example of a web service is to be found amongst the many good samples distributed with Dyalog APL.

C:\Program Files\Dyalog\Dyalog APL 11.0\Samples\asp.net\webservices\eg1.asmx

```
<%@ WebService Language="Dyalog" Class="APLExample" %>

:Class APLExample: System.Web.Services.WebService
:Using System
:Access public
     ∇ R←Add arg
     :Access WebMethod
     :Signature Int32←Add Int32 arg1,Int32 arg2
      R←+/arg
     ∇
:EndClass
```

The precise interpretation of this script is deferred until Module 20.

19.3.3.1 Call this service in your browser by typing http://localhost/dyalog.net/eg1.asmx?op=Add.

Here is a reminder summary of the sorts of files we have met so far, and their possible contents. Note the use of the symbol § to indicate the end of a control statement or tag.

Note double and triple dot single character symbols for implied missing code, and other shorthand:

- ⍺⍺… must end with (implicit or explicit) ⋄, ⍝ or §.
- ⍺⍺..⍵⍵ can march right through ( ) ↲ or ∇'s but not ⋄ ⍝ or §.
- ..⍵⍵ can have been through ( ) ↲ ∇'s ⋄ or ⍝ but not §.

File Type	Script Summary {optional}
*.html	<html>↲…</html>
*.asapx	<script…⍝∇…§<html>…§
*.aspx	<%@Page.. *.apl"%><html>… §
*.aspx	<%@Register..<html>..<dyalog: …§… ⍝ for custom control
*.apl	{:Namespace…⍝}:Class…⍝⎕USING…⍝∇…⍝ :Property…§
*.apl	:Namespace…⍝⎕LX…⍝∇…⍝..⎕WC… ⍝ for console appl
*.aspx	<%@Page Language="dyalog" Inherits="ñ" src="*.dws"%>↲{<html>..}
*.dws	¢ containing :Class…⍝:Using…⍝∇Page_Load…⍝..§
*.dws	#.ñ.⎕WC'NetType' 'Page' containing ∇Page_Load… and other exported methods…
*.asmx	<%@WebService..Class=..%>..⍝:Class..:Base…⍝…§
*.asax	<script…⍝∇App_Start…§
*.asmx	<%@WebService Class="ñ.¢"§ ⍝ call pre-defined .dll
*.dws	#.ñ.o.⎕WC'NetType' 'WebService' and exported web methods… ⍝ export to .dll

Orange rows are, perhaps, signposts to the way ahead. We shall explore in some detail the new APL semantics introduced for writing primitive APL classes in the next and final module of this course.

19.3.3.2 Please demand the final module 😐.