Interface APL+Win and .Net (C#)

Eric Lescasse

Lescasse Consulting
18 rue de la Belle Feuille
92100 Boulogne
France
Tel : 33 1 46 05 10 76
Fax : 33 1 46 04 60 23
eric@lescasse.com
http://www.lescasse.com

APL2000 User Conference
November 11, 2008
Table of Contents

Introduction ............................................................................................................................................................ 4
Small C# and Visual Studio Primer .......................................................................................................................... 5
  A simple C# method ............................................................................................................................................... 5
  A simple C# application in 18 easy steps .................................................................................................................. 6
Using .Net C# DLLs from APL+Win ........................................................................................................................ 16
  Transforming a C# DLL to be COM visible ........................................................................................................ 16
  Using the C# DLL from APL+Win ....................................................................................................................... 19
  Using NetAccess .................................................................................................................................................. 22
Developing a Client Server C# + APL+Win Application .......................................................................................... 35
  Introduction ....................................................................................................................................................... 35
  Requirements ....................................................................................................................................................... 36
    Registering APL+Win as a ActiveX Server ....................................................................................................... 36
    The APLServer DLL ......................................................................................................................................... 36
    Developing your first Client Server C# + APL+Win application .................................................................... 36
  Conclusion ......................................................................................................................................................... 52
Developing a Client Server ClickOnce C# + APL+Win Application ........................................................................ 57
  Introduction ....................................................................................................................................................... 57
  Principles ............................................................................................................................................................. 57
  Development Process ......................................................................................................................................... 57
  Debugging a ClickOnce application .................................................................................................................. 58
  Example .............................................................................................................................................................. 58
  Conclusion .......................................................................................................................................................... 64
Developing an ASP.Net Web Site using APL+Win in the Background ..................................................................... 65
  Introduction ....................................................................................................................................................... 65
  Examples ............................................................................................................................................................. 65
  ASP.Net Code ...................................................................................................................................................... 71
General Conclusion ............................................................................................................................................. 79
FAQ...........................................................................................................................................................................80

Why is it important to interface APL+Win and C#? .................................................................80
What benefit can an APL developer get from interfacing APL+Win and C#? .............................80
How difficult is it to interface APL+Win and C#? .................................................................80
How much should I learn about .Net and C#? .................................................................80
Where can I find help on learning .Net and C#? .................................................................80
In which ways can an APL developer interface to .Net C#? ..................................................81
What should the role of APL+Win and of C# be? ...............................................................81
Why C# and not VB.Net? .........................................................................................................81
What IDE (Integrated Development Environment) should I use to develop my APL+C# application? ...............82
How much does it cost? ............................................................................................................82
Should you use VS 2005 or VS 2008? ..................................................................................82
Is VS 2008 a secure development environment? ..................................................................82
I have a huge APL application: can I port it to C# + APL+Win? ............................................82
Will my C#+APL+Win application be difficult to debug? .....................................................83
If .Net C# is so good why do I still need APL at all? .............................................................83
Does interfacing C# and APL+Win really works in real life? ................................................83
**Introduction**

This presentation will demonstrate various ways by which one can interface APL+Win and C# (.Net).

It will do so through a number of demos and examples.

Basically, we will be looking at the following:

1. How to develop a C# DLL and use it from APL+Win (using NetAccess)
2. How to develop your application User Interface in C# and how to use APL in the background?
3. How to port your existing APL application to .Net using C# and APL+Win?
4. How to publish your APL + C# application as a Client Server ClickOnce Internet application?
5. How to write ASP.Net Web Sites using APL+Win in the background?

These are only a few of the things you can do with C# and APL+Win.

I know many of you are eager to learn more about .Net and wish they could use it while still benefiting from the tremendous power of APL. This talk is exactly about that. I have been studying .Net and C# for the last 3 years and got myself totally involved in this process to the extent that I can now do as much in C# as I can in APL.

What I have discovered is that those 2 languages interface “perfectly” well: and when I say “perfectly”, it is really “perfectly”. There is no compromise, the interface speed is just great and for example, you can easily exchange any piece of data between the 2 languages, even the most complex nested arrays as well as call any APL+Win function from C# as if it were inside your C# application!

At the end of this paper you’ll find a little FAQ trying to answer a few questions that you may have in mind. If you have other questions, please feel free to ask them.
Small C# and Visual Studio Primer

In order for you to better follow the demos and code samples included in this document, let’s introduce C# and Visual Studio 2008 through a sample application.

A simple C# method

Here is a little example to make you become acquainted with C#.

Here is how you would translate the +/- APL operator in C#:

```csharp
public double PlusRed(double[] values)
{
    double result = 0;
    for (int i = 0; i < values.Length; i++)
        result += values[i];
    return result;
}
```

Let’s explain the PlusRed C# method a bit: maybe the best way to explain it to an APL developer is to show its literal translation to APL:

```
\( \text{result} \text{PlusRed values;i} \\
[1] \text{result}=0 \\
[2] :\text{for i :in } \text{APLvalues} \\
[3] \text{result }+=\text{ result }+ \text{values}[i] \\
[4] :\text{endfor} \\
\)
```

First, PlusRed is the name of the method (similar to a function name in an APL workspace).

The PlusRed method accepts one argument which name is values but which is declared to be a vector ( "array" in C#) of "double". "double" is the data type and is the same as what we call floating point in APL; "[]" following "double" means that it is a one dimensional array. In C#, you must declare the data type of a variable before you can use the variable.

Similarly, the term "double" preceding PlusRed means that the result of the PlusRed method is a "double".

The body of the method is everything included within the curly braces:

```csharp
double result = 0;
for (int i = 0; i < values.Length; i++)
    result += values[i];
return result;
```

We first declare a variable called "result" to be a "double" and initialize it to 0.

---

1 You can of course choose any other variable name than "result", but I got the habit of calling my methods results "result" since it makes code a bit easier to read, understand and maintain
Then, there is a “for” loop in which we increment “result” by “values[i]”, result += values[i];
being equivalent to: result = result + values[i];

Finally, we return the content of the result variable as the method result.

This little example may let you guess 2 things:

1. First: the C# syntax is simple and easy to understand
2. Second: it should be easy enough to rewrite most APL primitives as C# methods which you could directly use in your C# programs (we’ll see more of this later on)

A simple C# application in 18 easy steps
We now need to test our C# method: in order to do so, we will create a C#.Net application using Visual Studio 2008. We won’t have room to explain everything in detail here, but will just give you the basic steps and a few important information so that you get a flavor of how to create a C#.Net application.

Proceed as follows:

1. Start Visual Studio 2008 SP1
2. Select: File / New / Project ...
3. Select Visual C# as the Project Type and Class Library as the Visual Studio Installed Template
4. Enter: APL2000.Conf2008 as the project name

If possible, always use the latest version of Visual Studio (as of August 27, 2008, it is Visual Studio 2008 SP1)
Alternatively you might want to use Visual Studio 2008 Express (or Visual C# 2008 Express which is equivalent) which is a free, somewhat simplified, version of Visual Studio 2008. The Express version is good enough to write small to medium size .Net applications. Note that, if you decide to use the Express version, dialog boxes are sometimes a little different than those displayed in this document, but you should be able to easily adapt.

Selecting Class Library means you will create a DLL
5. Click OK
This creates a new .Net C# project called APL2000.Conf2008 including one class, by default named Class1. Visual Studio creates the code for your project and displays it:

6. Copy/Paste the PlusRed method code into Class1
If you have not made any typo, the project should compile ok and the status bar should display: Build succeeded.
Congratulations: you have created your first C# DLL!
You can find it on your hard disk in the following directory:

C:\Documents and Settings\Eric Lescasse\My Documents\Visual Studio 2008\Projects\APL2000.Conf2008\APL2000.Conf2008\bin\Debug⁵

(replace Eric Lescasse by your User Name) and the DLL is named:


But how can we test our DLL? A DLL is a non visual piece of code which may contain methods (similar to APL functions), properties and fields (similar to APL variables): if you want, a DLL is a bit like an APL workspace without any User Interface code. So, we now need to add a Form to our Solution in order to be able to test our DLL.

8. Right Click on Solution ‘APL2000.Conf2008’ in Solution Explorer and select Add / New project...

9. In the Add New Project dialog, be sure to select Windows Forms Application this time and Name the Project: APL2000.Conf2008.Test

A new tab gets displayed containing the form you just added, in Design mode.

---

⁴ Note that a Solution in Visual Studio terminology is a collection of Projects. For now we have only one Project in our Solution, called APL2000.Conf2008 (same name as the Solution, by default), but we could and will add another Project to this Solution.

⁵ This is assuming you are running Windows XP and also, replace « Eric Lescasse » by your User Name.
10. Drag 2 Label objects, a TextBox object and a Button object onto the Form1 form and position them as follows (also reduce the form height a bit):

```
  Form1

  label1
     button1
   label2
```

11. Click on label1 so that it is selected and change its Text property (in the Properties pane at the bottom right of VS 2008) to: Values
This change gets immediately reflected in the form.
Click on button1 and change its Text property to: Compute
Click on label2 and change its Text property to: Result:
Also change its Font Size property to 14 and double click on False, next to Bold, to change it to True

---

6 Similar to an Edit object in APL+Win
7 Similar to the caption property of a Label in APL+Win
8 To change the Font Size, click on the + to the left of Font in the Properties pane and locate the Size sub-property
Your **Form** should now look as follows:

![Form Image]

We will use this form to enter values in the **TextBox** and then will display the result of running **PlusRed** in the **Result** label.

12. Right click on **APL2000.Conf2008.Test** in Solution Explorer and select **Set as StartUp Project**

   - The **Add Reference** dialog takes a few seconds to get displayed.
   - Select the **Browse** tab, and navigate to the **APL2008.Conf2008 / Bin / Debug** directory.
   - Double click on the **APL2000.Conf2008.DLL**

It’s now time to write some code in order to have a real functional application. We need to handle a **Click** event on the **Compute** button.

14. Double click on the **Compute** button.
   - This creates an empty event handler for the **Click** event on the **Compute** button and displays it in the **Form1.cs** tab:

---

9 This informs Visual Studio 2008 that clicking on the **Form** toolbar button should start the **Form** and not try to start the **DLL**: a DLL is not a Windows application and cannot be "started" from Visual Studio. That’s why you need a **Form** to be able to test your **DLL**.


11 Do not confuse the button **Text** property (i.e. “Compute”) with its **Name** property (which currently is **button1** i.e. the default Name allocated by Visual Studio to the Button when you added it to the Form).
Before you can use the APL2000.Conf2008 Class1 class in your Form you need to do 3 things.

Add the following using statement to the top of Form1.cs:

using APL2000.Conf2008;\(^{12}\)

Add the following statement just above public Form1():

private Class1 apl;\(^{13}\)

Add the following statement just below InitializeComponent():

apl = new Class1();\(^{14}\)

---

\(^{12}\) This statement informs Visual Studio that we will be using classes contained in the APL2000.Conf2008.DLL (namely: Class1)

\(^{13}\) It is very important to understand this statement: since it is in the Class1 class, but outside of any method, it declares the apl variable to be a global variable for the class, i.e. we will be able to use it in any method in the class. Had we declared the apl variable within a method it would have been local to this method and not available to other methods in the class. Note that the apl variable (called "field" in C# terminology) is declared to be of type Class1. Because we have added the using APL2000.Conf2008 statement, we do not need to prefix Class1 with APL2000.Conf2008. Without the using APL2000.Conf2008 statement we would have had to use the fully qualified APL2000.Conf2008.Class1 name

\(^{14}\) The previous instruction: private Class1 apl; was only a type declaration: it was not an instantiation of the apl object. Before we can access members of the Class1 apl object (namely its PlusRed method), we need to create an instance of the apl object. This is done with the apl = new Class1(); instruction in the Form1 Constructor.
Your code should now look like this:

```csharp
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

{
    public partial class Form1 : Form
    {
        private Class1 apl;
        public Form1()
        {
            InitializeComponent();
            apl = new Class1();
        }

        private void button1_Click(object sender, EventArgs e)
        {
        }
    }
}
```

We are now ready to write code for our application, in the `button1_Click` event handler. We need to read the content of the `textBox1` TextBox: this is supposed to be a list of numbers separated by blanks. In APL we would use the `–` (Execute) primitive to convert this character string to numbers. We need to perform the same kind of transformation in C#: so why not add a new method called `Execute` to our APL2000.Conf2008 DLL?

16. Add the following method to the APL2000.Conf2008 Class1 class:

```csharp
public class Class1
{
    public double PlusRed(double[] values)
    {
        double result = 0;
        for (int i = 0; i < values.Length; i++)
            result += values[i];
        return result;
    }

    public double[] Execute(string text)
    {
        string[] values = text.Split(new char[] { ' ' });
        double[] result = new double[values.Length];
        for (int i = 0; i < values.Length; i++)
            result[i] = Double.Parse(values[i]);
        return result;
    }
}
```
This method is interesting in that it allows me to introduce the .Net Framework. The .Net Framework is the set of thousands of classes built in Microsoft.Net and available to any C# developer. It is a gigantic\textsuperscript{15} collection of objects which you can use at any time in your applications and which basically reduce the amount of code you’ve got to write by 95% or so. That’s why .Net is so important for an APL+Win user: we can no longer afford not be able to tap into the .Net Framework power. This is certainly the most important thing an APL+Win developer needs nowadays.

The Execute argument called text is a string. We need to split it into an array of strings each containing one number, using the space character as the delimiter. Everything in C# is an object, even a single number, or a string. So we can type a dot after the text variable name to see with Intellisense which methods are available for a string object. And here we can see the Split method which does exactly what we need.

Intellisense tells us what the argument and result types are for any given method and the fundamental idea of C# programming is that you MUST always perfectly conform to these types.

For example, here is the Intellisense information displayed about the Split method for a string:

\[
\text{string[]} \text{string.Split(text, separator) \{ + 5 overload(s)\}} \\
\text{Returns a string array that contains the substrings in this instance that are delimited by elements of a specified Unicode character array.}
\]

This means that the result of the Split method MUST be declared as an array of strings (string[]) and that its argument must be an array of chars (char[]). As long as you 100% respect the types in C#, you’ll be safe. Hence the instruction:

\[
\text{string[]} \text{values = text.Split(' ') \{ \text{' ' \}};}
\]

Similarly, we are using another .Net Framework method in the Execute method and this is Double.Parse which is a static\textsuperscript{16} method of the Double class. This method converts a character string to a double, which is exactly what we need to do here.

17. Right click on APL2000.Conf2008 and select Rebuild to recompile\textsuperscript{17} the APL2000.Conf2008 project

18. Now for the final step in our application: writing the button1_Click event handler.

\textsuperscript{15} Just to give you an idea of how gigantic and powerful the .Net Framework is, there are 40513 types (or classes) in the .Net Framework 3.5 SP1 and 386790 methods. 1393 new public methods were added just for the .Net framework Service Pack 1. This does not mean the .Net Framework is complex to use: paradoxical as it may seem, it is in most cases pretty easy to use, once you understand the basics.

\textsuperscript{16} A static method is a method which you use on the Class itself, not on an instance of the Class.

\textsuperscript{17} We need to recompile the APL2000.Conf2008 project so that are change (the newly added Execute method) be included in the APL2000.Conf2008.DLL which we are going to use from the APL2000.Conf2008.Test project, otherwise it would be the last compiled version of APL2000.Conf2008.DLL which would be used and which did not yet contain the Execute method.
Add the following code to the button1_Click event handler in Form1.cs:

```csharp
private void button1_Click(object sender, EventArgs e)
{
    double[] values = apl.Execute(textBox1.Text);
    double result = apl.PlusRed(values);
    label2.Text = "Result: " + result;
}
```

In this event handler remember that we are using apl which is an instance of our APL2000.Conf2008.Class1 class.

We are first calling its Execute method to convert the values entered by the user in the textBox1 TextBox to a numeric vector of doubles. We then call its PlusRed method, passing this vector of doubles as an argument, to compute its sum. And we finally display the result as a character string in the label2 Label.

Pretty easy, isn’t it?

Those of you who are APL one liners, will certainly appreciate the one line version of our little handler (yes, we can also be one liners in C#!):

```csharp
private void button1_Click(object sender, EventArgs e)
{
    //double[] values = apl.Execute(textBox1.Text);
    //double result = apl.PlusRed(values);
    //label2.Text = "Result: " + result;
    label2.Text = "Result: " + apl.PlusRed(apl.Execute(textBox1.Text));
}
```

I have commented\(^{18}\) the previous lines for an easier comparison of the 2 versions.

Why has it been so simple to write the button1_Click handlers?

This is because of our Object Oriented approach to this application: we have taken the time to set up a Class1 object and to write the methods we would need to make our Form1.cs development easy.

Moreover, Class1 being in an external DLL (i.e. external to our APL2000.Conf2008.Test application) we will be able to reuse it in the future, as often as we need it. For example, every time we will have to convert a TextBox content to a numeric vector or every time we will have to compute the sum of a numeric vector, we can reference the APL2000.Conf2008.DLL in our application and reuse the Class1 class.

Now there’s no interest in writing an application if it’s not for using it.

---

\(^{18}\) Comments in C# start with 2 slashes
Let’s start and use our application.

1. Click on the toolbar button
2. Enter: 10 20 30 in the TextBox
3. Click the Compute button

Here is what you get:

Result: 60
Using .Net C# DLLs from APL+Win

You have now seen how to create a C# DLL.

Of course, you still have to learn more about C# and the .Net Framework to be able to write more complex and useful DLLs, but the good news is that there are myriads of books and Web Sites which can help you in that task. In any case, if you want to write your own C# DLLs and use them from APL+Win, you’ll have to invest a bit of your time in learning C# and the .Net Framework.

If you get decided and make the necessary initial effort, you’ll quickly find out it is much easier than what you might have thought at first.

What is the difference between a standard C# DLL and one that can be used from APL+Win?

The main difference is that you must make your DLL a COM Interop DLL:

1. You must change some DLL properties before compiling it (steps 1 to 6 below)
2. You must add a number of statements to your DLL (steps 7 and above below)

Transforming a C# DLL to be COM visible

Let’s transform our APL2000.Conf2008 DLL to make it a COM Interop DLL:

   The APL2000.Conf2008 tab gets opened:

   ![Properties window](image)

2. Click the Assembly Information... button
3. Check the Make assembly COM-Visible check box at the bottom of this dialog
4. Click OK
5. In the APL2000.Conf2008 tab, click the Build tab
6. At the bottom of this tab, check the Register for COM interop check box

7. Next, change the Class1.cs code as follows (adding all the highlighted statements):

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;

```
The added lines are complex code which you should not be interested in studying in much detail and indeed there is a way to generate them automatically (see NetAccess later on in this document).

Let’s just say that, in order to transform a C# DLL into a COM Interop DLL you must create an Interface in your DLL. An Interface is a section of code where you declare the prototypes of all the public properties and methods included in your DLL. The Interface name is the class name Class1 prefixed with an I, hence IClass1.

Next you tell Class1 that it must inherit from the Interface, by adding : IClass1 to the Class1 declaration.

The GUIDs are unique strings which you can generate using the VS 2008 Tools / Create GUID dialog. You can use any new GUID generated by this dialog. Be sure to embed the generated GUIDs in double quotes and to remove the curly braces.

The sections of code embedded between brackets such as:

[ClassInterface(ClassInterfaceType.None)]

are called Attributes in C# and are classes used to store information about the program constructs.
8. Right click **APL2000.Conf2008** in **Solution Explorer** and select **Rebuild** to recompile the DLL

You now have a COM Interop DLL available for use by APL+Win (or any other development system able to use ActiveX)!

**Using the C# DLL from APL+Win**

It's now pretty easy to use our C# DLL from APL+Win, using **ŒWI**.

1. Start **APL+Win** (any version ≥ 4)
2. Create an instance of the **APL2000.Conf2008.Class1** class:

   ```
   'apl'Œwi'Create' 'APL2000.Conf2008.Class1'
   ```

3. See what methods are available:

   ```
   'apl'Œwi'methods'
   Close Create Defers Delete EnumEnd EnumNext EnumStart Event Exec Info Modify New Open Ref Send Set SetLinks XExecute XPlusRed
   ```

   As you can see, there is a bunch of methods which all APL+Win objects share, plus 2 new ActiveX methods: **XExecute** and **XPlusRed**

4. Use these methods:

   ```
   'apl'Œwi'Execute' '10 20 30'
   10 20 30
   +/'apl'Œwi'Execute' '10 20 30'
   60
   'apl'Œwi'PlusRed'(10 20 30)
   60
   ```

   Note that you would get an error if you tried:

   ```
   'apl'Œwi'PlusRed'10 20 30
   LENGTH ERROR: Wrong number of arguments
   'apl'Œwi'PlusRed'10 20 30
   ^
   ```

   That makes sense since we defined the **PlusRed** method in C# as having a single argument which was a vector of doubles. Therefore, you cannot pass 3 arguments: 10, 20 and 30 to the **PlusRed** method as we tried above.

   You also see that doing a wrong call to a C# DLL method does not hang your system but simply reports a standard APL error which you can easily trap with **Œelx** if you need so.
Now, what about calling `PlusRed` with a very large argument (one million values, i.e. one million loops in the C# code):

```
Œpp+17
 'apl' 'dw' 'PlusRed'(1000000)
500000500000
```

Well: if you try this, you’ll see that the answer is instantaneous.

OK: I know this example is not terrific: why use a C# DLL to do things that APL+Win does with less code (+/) and a bit faster?

But what if you had to do things like the following:

Write a method computing the week of year number corresponding to a given date; the method must work in all countries (note: some countries use **Sunday** as the first day of week, others use **Monday**; moreover different countries use different rules [FirstDay, FirstFullWeek, FirstFourDayWeek] to determine which is the first week).

You sure could do that in APL, however, there is a fairly high chance it would take you “a good while” before you get the right results for every country.

Let’s add a **WeekOfYear** C# method to our DLL.

5. If APL+Win is still active, close your APL+Win Session

6. Add the following method to the **Class1** class

```csharp
public int WeekOfYear(int year, int month, int day, string culture)
{
    CultureInfo ci = new CultureInfo(culture);
    DateTime dt = new DateTime(year, month, day);
    DayOfWeek dow = ci.DateTimeFormat.FirstDayOfWeek;
    CalendarWeekRule cwr = ci.DateTimeFormat.CalendarWeekRule;
    return ci.Calendar.GetWeekOfYear(dt, cwr, dow);
}
```

This method is a good example of using the **.Net Framework**.

A **CultureInfo** object is first instantiated, using the culture passed as an argument (example: “en-US”)

Then a **DateTime** object is instantiated using the year, month and day passed as arguments. Then a **DayOfWeek** object corresponding to the first day of week for the specified culture is created (it will sometimes correspond to a **Sunday**, sometimes to a **Monday**, etc.)

---

19 If you don’t close your APL Session, you won’t be able to recompile the DLL, because it is “currently in use” in your APL Session
Then the `CalendarWeekRule` Enum member corresponding to the specified culture is retrieved and saved in the `cwr` variable.

Then the `GetWeekOfYear` method of the `Calendar` object corresponding to the specified culture is run, using the objects retrieved above.

It returns the correct week number corresponding to the specified date for the specified culture.

This example is typical of the .Net Framework where you need to use various classes and combine them the right way to achieve what you want. It may seem complex at first sight, but the good news is that you can always Google for hundreds of examples corresponding to what you’re after. You almost always find the right example for your problem or at least find inspiring examples that quickly lead you to the right solution.

7. Add the following prototype to the `IClass1` Interface:

```csharp
int WeekOfYear(int year, int month, int day, string culture);
```

8. Recompile the DLL

We are now ready to use it from APL.

9. Start APL+Win

10. Create an instance of the DLL

    `'apl'Dwi'Create' 'APL2000.Conf2008.Class1`

11. Use the `WeekOfYear` method

```apl'
apl'Dwi'WeekOfYear' 2005 1 1 'ur-PK' © Pakistan (implements FirstFullWeek)
apl'Dwi'WeekOfYear' 2005 1 1 'it-IT' © Italy (implements FirstFourDayWeek)
apl'Dwi'WeekOfYear' 2005 1 2 'it-IT'
apl'Dwi'WeekOfYear' 2005 1 3 'it-IT'
apl'Dwi'WeekOfYear' 2005 1 1 'en-US' © United States (implements FirstDay)
```
Using NetAccess

**NetAccess** is an APL product developed by **Lescasse Consulting**: it is a C# Code Generator which allows you to auto-generate the code for your COM Interop C# DLLs.

Let’s create another DLL with a few simple properties and methods.

1. Start **APL+Win** and load the **NetAccess.W3** workspace
   The NetAccess 2.1 main form is displayed
2. Enter: **LescasseConsulting.FileSystem** in the **NameSpace** field
3. Enter: **Watcher** in the **Class** field
4. Click the **Generate** button
   The NetAccess 2.1 form should look like this:

   ![NetAccess 2.1 Form](image)

   NetAccess has generated the code for your DLL, but for now, the DLL does not include any property, method or event. We need to add some of those to make our DLL do something.

5. Click on the **Properties** button
6. Define 3 properties as follows:

   ```
   using System;
   using System.Collections.Generic;
   using System.ComponentModel;
   using System.Data;
   using System.Text;
   using System.Runtime.InteropServices;

   namespace LescasseConsulting.FileSystem
   {
   [Guid("C8840D77-F31F-4AP9-56E3-C19492B6B626")]
   [InterfaceType(InterfaceType.Sequential)]
   public interface IWatcher
   {
   }

   [Guid("091F3DA-2FX6-8961-F0348C96F6B6")]
   [InterfaceType(InterfaceType.Sequential)]
   [ProgId("LescasseConsulting.FileSystem.Watcher")]
   public class Watcher : IWatcher
   {
   public Watcher()
   {
   }
   }
   }
   ```

---

*Since this document was written a new NetAccess version 2.2 has been released!*
7. Click on the C# Code button
As you can see, the C# code for your properties has been automatically added to the DLL:

```csharp
7. Click on the C# Code button
As you can see, the C# code for your properties has been automatically added to the DLL:
```

8. Click on the Events button
9. Define the following event
10. Click the **C# Code** button to see the automatically generated C# code added to the DLL:


12. Click: **File / New / Project...**

13. Create a new **Class Library** project named **LescasseConsulting.FileSystem**
14. Click OK
You should get the following screen:

![Screen capture showing the .NET IDE with a class and its contents.

15. Right click on Class1.cs in Solution explorer and select Rename
16. Rename it: Watcher.cs
17. Answer Yes to the following question:

![Confirmation dialog box asking if you want to rename all references.

18. Replace the whole C# code contained in the Watcher.cs file with the code generated by NetAccess

![Code comparison dialog box showing before and after code.

19. Press Ctrl+S to save the changes to Watcher.cs
20. Right click on Solution ‘LescasseConsulting.FileSystem’ and select Build Solution to compile the DLL
   The status bar displays Build succeeded indicating that our C# generated code did not have any flaws.
We still need to declare that our DLL is a COM Interop one.

21. Double click **Properties** under *LescasseConsulting.FileSystem* in *Solution Explorer*

The *LescasseConsulting.FileSystem* tab gets opened:

![Image of Solution Explorer with LescasseConsulting.FileSystem tab opened]

22. Click the **Assembly Information...** button

23. Check the **Make assembly COM-Visible** check box at the bottom of this dialog

![Image of Assembly Information dialog]

24. Click **OK**

25. In the *LescasseConsulting.FileSystem* tab, click the **Build** tab

26. At the bottom of this tab, check the **Register for COM interop** check box
At this stage we can compile the DLL again and try using it from APL+Win.

Here is a short APL Session showing how to create an APL instance of the LescasseConsulting.FileSystem.Watcher object.

However our C# DLL does not yet include any really util code.

27. Let’s include some useful code to our DLL.

First, let’s add the following statement at the top of the DLL:

```csharp
using System.IO;
```

We need to tell VS2008 that we are going to use a class (FileSystemWatcher) contained in the System.IO namespace.

Then, we need to create an instance of a C# FileSystemWatcher object in the DLL Constructor.
Locate the following code, which represents the Constructor, in the DLL:

```csharp
public Watcher()
{
}
```

and change it to:

```csharp
private FileSystemWatcher fsw;

public Watcher()
{
    fsw = new FileSystemWatcher();
}
```

Note that the FileSystemWatcher object (fsw) declaration is made outside of the Constructor so this object is a global variable for the DLL.

Now make the fsw object subscribe to the Changed event. To do that properly in VS2008, type the following under fsw = new FileSystemWatcher(); in the Watcher Constructor:

```csharp
fsw.Changed +=
```

Just after typing the = sign, press Tab to generate the remaining code in this line.

Then press Tab again to generate the event handler. You should now see the following code:

```csharp
public Watcher()
{
    fsw = new FileSystemWatcher();
    fsw.Changed += new FileSystemEventHandler(fsw_Changed);
}

void fsw_Changed(object sender, FileSystemEventArgs e)
{
    throw new NotImplementedException();
}
```

We are almost done.
We now need to fire our own onChanged event from within the fsw_Changed event so that APL could be notified of the file changes having occurred in the file system.
28. Add the following code (highlighted) to the fsw_Changed event handler:

```csharp
void fsw_Changed(object sender, FileSystemEventArgs e)
{
    onChanged(e.FullPath, e.ChangeType.ToString());
}
```

This code fires our onChanged event with 2 arguments:

- `e.FullPath` will contain the full path name of the changed file
- `e.ChangeType` will contain the type of file change that occurred

However, the return type of `e.ChangeType` being a `WatcherChangeTypes` object in C#, we need to transform this to a string, which is easily done with the `ToString()` method.

We also need to write the C# code for our 3 properties: `enabled`, `path` and `subdirs`.

This will be simplistic since these properties are just cover properties for similar `FileSystemWatcher` ones in C#.

Change the code of these 3 properties to the following:

```csharp
private bool _subdirs;
public bool subdirs
{
    get { return fsw.IncludeSubdirectories; }
    set { fsw.IncludeSubdirectories = value; }
}

private string _path;
public string path
{
    get { return fsw.Path; }
    set { fsw.Path = value; }
}

private bool _enabled;
public bool enabled
{
    get { return fsw.EnableRaisingEvents; }
    set { fsw.EnableRaisingEvents = value; }
}
```

You may also now remove the `_subdirs`, `_path` and `_enabled` fields since they are no longer use in the DLL. Remove the following lines:

```csharp
private bool _subdirs;
private string _path;
private bool _enabled;
```
29. Re-compile the DLL by right clicking on Solution ‘LescasseConsulting.FileSystem’ and selecting Rebuild Solution.

Note: if you get a compile error telling you that the DLL is in use, it is because you still have an APL Session opened where you created an instance of the DLL, earlier. Just exit APL and try compiling again. Everything should be fine now.

30. Let’s use our DLL from APL.
Start APL+Win

Create an instance of the C# DLL:

'fsw'Œwi'Create' 'LescasseConsulting.FileSystem.Watcher'

Indicate that we want to watch file changes in the C:\ directory:

'fsw'Œwi'path' 'C:\'

Indicate that we want to watch all subdirectories of this directory:

'fsw'Œwi'subdirs'1

Define the XonChanged event:

'fsw'Œwi'*onXonChanged' 'Œ"Œwarg'

Note the necessity to add the on prefix before the X in the XonChanged event.

Now enable our LescasseConsulting.FileSystem.Watcher object:

'fsw'Œwi'enabled'1

As soon as you do that you get the following kind of output in your APL Session (captured from my APL Session):

C:\Documents and Settings\All Users\Application Data\Microsoft\Search\Data\Applications\Windows\Projects\SystemIndex\indexer\CFiles\00010039.wid Changed
C:\Documents and Settings\All Users\Application Data\Microsoft\Search\Data\Applications\Windows\Projects\SystemIndex\indexer\CFiles\00010039.ci Changed
C:\Documents and Settings\All Users\Application Data\Microsoft\Search\Data\Applications\Windows\Projects\SystemIndex\indexer\CFiles Changed
C:\Documents and Settings\All Users\Application Data\Microsoft\Search\Data\Applications\Windows\Projects\SystemIndex\indexer\CFiles\00010039.wid Changed
C:\Documents and Settings\All Users\Application Data\Microsoft\Search\Data\Applications\Windows\Projects\SystemIndex\indexer\CFiles\00010039.ci Changed
C:\Documents and Settings\All Users\Application Data\Microsoft\Search\Data\Applications\Windows\Projects\SystemIndex\indexer\CFiles Changed
As you can see, there are lots of file changes occurring on our computers at all times, even if we aren’t using our computer!

We are not really interested in all these changes, so let’s watch only one specific directory: change the path to be C:\TEST

```
fsw 'Dwi' 'path' 'C:\TEST'
```

But, let’s create a new file in this directory, then rename it:

```
c:\test\test' 'fcreate 1
C:\\Test Changed
C:\\Test\TEST.sf Changed
```
```
funtie 1
'c:\test\test' 'ftie 1
'c:\test\test2' 'rename 1
```
```
c:\test\TEST.sf Changed
```
```
c:\test\test2' 'ferase 1
```
```
c:\test\TEST2.sf Changed
```
The reason we are getting the word Changed all the time (instead of Created, Renamed, Deleted) is that we only handled the Changed event.

Change the constructor as follows:

```csharp
public Watcher()
{
    fsw = new FileSystemWatcher();
    fsw.Changed += new FileSystemEventHandler(fsw_Changed);
    fsw.Created += new FileSystemEventHandler(fsw_Changed);
    fsw.Deleted += new FileSystemEventHandler(fsw_Changed);
    fsw.Renamed += new RenamedEventHandler(fsw_Changed);
}
```

to handle more FileSystemWatcher events. Note that all of these 4 events share the same event handler: fsw_Changed. This way we do not have to create any other event handler and we are ready to go.

31. Exit APL
32. Recompile the DLL
33. Start APL and try the following:

```apl
'fsw'Œwi'Create' 'LescasseConsulting.FileSystem.Watcher'
fsw
'fsw'Œwi'subdirs'1
'fsw'Œwi'path' 'c:\test'
'fsw'Œwi'enabled'1
'fsw'Œwi'*onXonChanged' 'Œ"Œwarg'
'
'c:\test\test'Œfcreate 1
 c:\test\TEST.sf Created
 c:\test\TEST.sf Changed

Œfuntie 1
'c:\test\test'Œftie 1
'c:\test\test2'Œfrename 1
 c:\test\TEST.sf Changed
 c:\test\TEST2.sf Renamed

(\10)Œfappend 1
 1
 c:\test\TEST2.sf Changed

'c:\test\test2'Œf erase 1
 c:\test\TEST2.sf Deleted
```

As you see we get informed about any change made to any file in the C:\TEST directory.
Here is the complete code of our DLL:

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;

namespace LescasseConsulting.FileSystem
{
    public delegate void onChangedEvent(string path, string changetypes);

    [Guid("C884D77F-221F-4AF9-96E3-C1F09FE43862")]
    [InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
    public interface IWatcher
    {
        bool enabled { get; set; }
        string path { get; set; }
        bool subdirs { get; set; }
    }

    [Guid("94AD4937-BBAE-49A5-9EC1-3C03D766F1E1")]
    [InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
    public interface IWatcherEvents
    {
        [DispId(91)]
        void onChanged(string path, string changetypes);
    }

    [Guid("9313CE34-E93D-4010-BB81-FB7428C39FD8")]
    [ComSourceInterfaces(typeof(IWatcherEvents))]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("LescasseConsulting.FileSystem.Watcher")]  
    public class Watcher : IWatcher
    {
        public event onChangedEvent onChanged;

        public bool subdirs
        {
            get { return fsw.IncludeSubdirectories; }
            set { fsw.IncludeSubdirectories = value; }
        }

        public string path
        {
            get { return fsw.Path; }
            set { fsw.Path = value; }
        }

        public bool enabled
        {
            get { return fsw.EnableRaisingEvents; }
            set { fsw.EnableRaisingEvents = value; }
        }

        private FileSystemWatcher fsw;
        public Watcher()
        {
```

To sum up, with only 62 lines of C# code we have created a COM Interop DLL, usable from APL+Win, to let us watch any changes made to files within any directory of our system.
Developing a Client Server C# + APL+Win Application

Introduction
Another possible way of interfacing C# and APL is to write Client Server C# + APL+Win applications.

In this case, C# and APL both have a very specific role:

- C# is the Client and APL the Server
- C# handles the User Interface and APL handles calculations, files, etc.

Basically, your application becomes a .Net application.

You develop your User Interface using Visual Studio and C#.

When time comes to write event handlers, you decide if you want to write the event handler in C# or if you want to sub-contract the event handler to APL. You can also use a mix of the 2.

This way of writing applications is very powerful and efficient because you use each software (C# and APL) for its best strength: C# and Visual Studio are unbeatable for writing User Interfaces and by far; APL is also hard to beat for its computational power, its flexible (colossal) component files, etc.

You must know that when writing applications this way:

1. You get a total access to your APL workspace from C# (variables, functions, system variables, system functions, commands, User Command Processor, etc.)
2. Calls into the APL workspace from C# are extremely easy
3. Calls into the APL workspace from C# are extremely fast: there is virtually “no” overhead at all involved.
4. You can exchange any kind of data between APL and C#, even the most complex nested arrays!

I am known to have always been fond of developing User Interfaces: this is the thing I like to do best. And I have enjoyed developing APL User Interfaces for years and years.

Having learnt how to do it with C# and Visual Studio, I think I will never again write any User Interface in APL. My clear choice now is to use C#/Visual Studio to develop User Interfaces and to use APL in the background for data processing, files, etc. In some cases I would use SQL Server instead of APL files, especially for multi user transactional applications.

Interfacing C# and APL this way has 2 other advantages:

1. It forces you to totally separate the User Interface from the Business logic of your application, something APL developers have never been taught to do, but which is a basic rule adopted by a vast majority of application developers all around the world
2. It allows you to easily transform your application to an Internet ClickOnce Client Server C# + APL application (more about this later)
Requirements

So what do you need to write a Client Server C# + APL+Win application?

Believe it or not, you just need to register APL+Win as an **ActiveX Server** and you need to use a short and simple little DLL.

Registering APL+Win as a ActiveX Server

Most often this has already been done when APL+Win was installed on your computer, if you answered Yes to the final installation questions.

But in any case, if you want to be sure your APL+Win is registered as an **ActiveX Server**, you can at any time open a **DOS Command Prompt** window and do the following:

- `regsvr32 path\aplwco.dll`
- `path\aplw.exe MyApp.ini /RegServer`

where *path* represents the directory where APL is installed on your computer (for example: *C:\APLWIN80*) and *MyApp.ini* represents your application .INI file.

The APLServer DLL

Here is how to create an APLServer C# DLL which will allow you to write Client Server C#+APL+Win application:

1. Start **Visual Studio 2008**
2. Select **File / New / Project...**
3. Select **Class Library** and enter **LescasseConsulting.APLServer** for the project name
4. Once the project is created, right click on **Class1.cs** in **Solution Explorer** and select **Rename**
5. Enter: **APLServer.cs**
6. Click **Yes** to answer the Visual Studio prompt
7. Select the whole code in **APLServer.cs** and replace it with the following code

   Request the code of the APLServer.cs DLL to Lescasse Consulting

8. In the Visual Studio toolbar select **Release** instead of **Debug** in the first combo box
9. Press **F6** to compile the DLL

Developing your first Client Server C# + APL+Win application

You can now proceed to develop your first Client Server C# + APL+Win application.
Creating the C# Client Application

Knowing the limited amount of time devoted to this lecture, our Client+Server C#+APL+Win application will be a simple one: a minimal human resources enterprise database.

2. Select File/New Project...
3. Select the Windows Forms Application template
4. Enter: APL2000.Conf2008App as the application Name
5. Check the Create directory for Solution check box
6. Click OK
7. In the Properties pane, change the StartPosition property to CenterScreen so that the form will get displayed in the center of the screen when the application starts

The C# application will be the Client application. It will contain the whole User Interface of our application.

The APL application will be the Server application. It will contain the Data Layer (i.e. the database and the necessary APL routines to use the database) as well as the Business Logic Layer.

Adding a Reference to the LescasseConsulting.APLServer DLL

Since we will want to use APL from C# we need to reference the LescasseConsulting.APLServer DLL in our application.

To do so:

8. Right click on References in Solution Explorer (just below Properties)
9. Select: Add Reference...
10. In the Add Reference dialog box, click on the Browse button
11. Navigate to the LescasseConsulting.APLServer\LescasseConsulting.APLServer\Bin\Release directory
12. Double click on the LescasseConsulting.APLServer.DLL file name (this adds a reference to this DLL to the project)
13. Double click on the Interop.APLW.DLL file name (this adds a reference to this DLL to the project)
14. Right click on the Form1 form and select View Code (alternatively, right click on Form1.cs in Solution Explorer and select View Code)

The Form1.cs code should look like this:

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace APL2000.Conf2008App {
```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
}

15. Add the following using clause at the bottom of the using clauses section:

using LescasseConsulting.AplServer;

16. Declare a private field called apl of type AplServer, just above the Form1() constructor

17. Create an instance of the AplServer apl object, pointing to the C:\aplwin\ele\conf2008.w3 workspace, in the Form1() constructor

That’s all it takes to be able to use anything you want in the APL+Win CONF2008.W3 workspace from your C# application!

The complete class code should now look like this:

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using LescasseConsulting.AplServer;

namespace APL2000.Conf2008App
{
    public partial class Form1 : Form
    {
        private AplServer apl;

        public Form1()
        {
            InitializeComponent();
            apl = new AplServer(@"C:\aplwin\ele\conf2008");
        }
    }
}

To understand this well with an analogy to APL, consider that the private AplServer apl; instruction role is only to declare a global variable called apl, but at this stage the apl variable is just declared, it is not yet instantiated and therefore you can’t yet use it.

The new AplServer(@"C:\aplwin\ele\conf2008"); instruction is the one which creates apl as an instance of the AplServer object, informing it to point to the CONF2008 workspace (note that you should not include the .W3 extension when passing an APL workspace as a parameter to the AplServer object constructor)
Because you have declared the apl variable (we say field in C#) outside of any method, it is a **global field** for the whole class and can therefore be used in any method of the Form1 class. Had we included the **private** AplServer apl; instruction within the Form1() method, the apl field would have been a local “variable” to the Form1() constructor and therefore would have been destroyed as soon as the Form1() constructor would have finished executing, just as a local APL variable stops existing as soon as the function it is localized in finishes executing.

**Creating the C# User Interface**

Let’s create our User Interface; we want to be able to navigate thru the records of our database:

18. Locate the BindingNavigator object in the VS2008 toolbox (it is located in the Data section)
19. Drag and drop a BindingNavigator object onto the form
20. Drag and drop 4 **Labels** to the form and change their captions to: Department, Name, FirstName and Birth Date
21. Drag and drop a ComboBox to the form and rename it: cbDept
22. Drag and drop 2 **TextBox** to the form and rename them txtName and txtFirstName
23. Finally drag and drop a DateTimePicker object to the form and rename it dtpBirthDate
24. Click on the **Form1** form and change its **Text** property (in the Properties pane) to:

**Human Resources Database**

25. Align controls and resize the form so that it looks like the following:

![Human Resources Database Interface](image)

We are done with the User Interface design of our application.

We now need to handle events, like clicks on the BindingNavigator buttons.

But before that, we need to populate the Department combo box with our Enterprise departments. We will do that in the **Form1 Load** event: this event is the first event which occurs just before the form gets displayed when you start the C# application so it’s a good place for initializing controls on the form.
26. Double click on a free area of the form
   (pay attention not to double click on any control)

   This creates an event handler in the Form1.ws code and displays it:

   ```csharp
   private void Form1_Load(object sender, EventArgs e)
   {
   }
   ```

27. Add the following code to this event handler:

   ```csharp
   object depts = apl.CallFn("GetDepts");
   cbDept.DataSource = (string[])depts;
   ```

   The first instruction calls the niladic GetDepts function in the CONF2008.W3 APL workspace and returns its result. The result of calling ANY function in an APL workspace is a C# object of type object. This type is the parent of all other C# types and as such can internally contain absolutely anything.

   We know that the GetDepts APL function returns a “vector of character vectors” (we call this an “array of strings” in C#).

   Therefore to really get an array of strings (noted: string[] in C#), we must cast the depts variable which is an object, to its real content which is a string[].

   This is done by using the following notation: (string[])depts

   We then simply assign this array of strings to the DataSource property of our cbDept ComboBox. That’s enough to populate the ComboBox.

   Isn’t this really easy?

   Note that we could have done all that in the following way:

   ```csharp
   string[] depts = (string[])apl.CallFn("GetDepts");
   cbDept.DataSource = depts;
   ```

   or even more simply (didn’t we all use to be one liners at one time as APL developers?):

   ```csharp
   cbDept.DataSource = (string[])apl.CallFn("GetDepts");
   ```

   avoiding having to create the depts variable at all.

   So, our Form1_Load event handler now looks like this:

   ```csharp
   private void Form1_Load(object sender, EventArgs e)
   {
   cbDept.DataSource = (string[])apl.CallFn("GetDepts");
   }
   ```

   Start the application and open the cbDept ComboBox:
As you can see the cbDept ComboBox has indeed been populated with departments extracted from an APL+Win Colossal File by an APL+Win function! Great!

Now we would also like to display the first database record when the form loads. We can do this by adding the following code to the Form1_Load event handler:

```csharp
object[] record1 = (object[])apl.CallFn("GetRecord", 1);
int dept = (int)record1[0] - 1;
string name = (string)record1[1];
string firstname = (string)record1[2];
int[] birthdate = (int[])record1[3];
cbDept.SelectedIndex = dept;
txtName.Text = name;
txtFirstName.Text = firstname;
dtpBirthDate.Value = new DateTime(birthdate[0], birthdate[1], birthdate[2]);
```

Let's explain this code:\(^{21}\): on the first line we call the GetRecord APL+Win function with a argument of 1 and capture its result in variable record1 which is declared to be an array of objects (object[]).

Here is the GetRecord APL function:

```
R=GetRecord number;tie;offset

1  #× R=GetRecord number -- Retrieves record <number> from database
2
3  tie=TieDb
4  :if number>0
5  :randif number<(2œŒcfsize tie)-DbOffset
6  R=œcfread tie,number+DbOffset
8  :else
9  R=’No record at this location!’
10  :endif
```

```
R=EncodeDate date

1  #× R=EncodeDate date -- Transforms a scalar (yyyyymmdd) date
2  #× into a 3-element vector (yyyy mm dd)
3  #× Note: Belongs to the Business Layer
4
5  R=c10000 100 100?date
```

We will explain a little later on that we have created an EncodeDate and DecodeDate subroutine to separate the Data Layer stuff from the Business Layer stuff.

\(^{21}\) As you can guess by reading this code, the index origin is always 0 in C#!
Remember that the result of calling an APL function is ALWAYS an object in C#, but that an object can contain anything, i.e. any other type or structure of types. Here, we know that the GetRecord APL function returns a nested vector containing integers and strings:

```
GetRecord 1
3 Swain Rex 1950 1 1
```

Therefore, since this vector is heterogeneous, we must declare it as an array of objects in C#.

To understand this well, just realize that each element of an array of objects is itself an object and as such can in turn contain anything. So, the first element is an object which contains an integer, the second and third element are objects which contain strings and the fourth element is an object which contains a vector of integers.

It then becomes easy to extract each element to its internal type, using the appropriate cast expression: this is what’s done in the following 4 instructions which create 4 local variables (dept, name, firstname and birthdate):

```
int dept = (int)record1[0] - 1;
string name = (string)record1[1];
string firstname = (string)record1[2];
int[] birthdate = (int[])record1[3];
```

It is then easy and simple to update the form controls with these 4 pieces of data:

```
cbDept.SelectedIndex = dept;
txtName.Text = name;
txtFirstName.Text = firstname;
dtpBirthDate.Value = new DateTime(birthdate[0], birthdate[1], birthdate[2]);
```

Note that the DateTimePicker Value property is a DateTime object: you can see that using Intellisense, by typing a dot (.) after dtpBirthDate and navigating to the Value property: here is what Intellisense displays:

```
DateTime DateTimePicker1.Value
Gets or sets the date/time value assigned to the control.
Exceptions:
    ArgumentException
```

In C#, as long as you respect types, you’ll be ok and when you understand that this is a strict universal rule, C# becomes pretty easy to understand.

Here, Intellisense tells us that a DateTimePicker Value property returns a DateTime object, therefore, if we want to set the dtpBirthDate Value property we must provide it with an instance of a DateTime object. This is exactly what the following instruction does:

```
dtpBirthDate.Value = new DateTime(birthdate[0], birthdate[1], birthdate[2]);
```
Finally, as we did before, we can easily avoid having to create temporary local variables and our Form1_Load handler can easily be refactored to the following code:

```csharp
private void Form1_Load(object sender, EventArgs e)
{
    cbDept.DataSource = (string[])apl.CallFn("GetDepts");
    object[] record1 = (object[])apl.CallFn("GetRecord", 1);
    cbDept.SelectedIndex = (int)record1[0] - 1;  //department
    txtName.Text = (string)record1[1];           //name
    txtFirstName.Text = (string)record1[2];      //first name
    dtpBirthDate.Value = new DateTime(
        ((int[])record1[3])[0],
        ((int[])record1[3])[1],
        ((int[])record1[3])[2]
    );
}
```

You may want to note a few things:

1. First, we have added comments to make our code more readable
2. Second, the last statement is written over 5 lines to also improve readability
3. Finally, the last statement comment is embedded within the statement itself (yes, this is possible in C#!)

28. We will need to display other database records using the BindingNavigator buttons.

Analyzing this quickly, we see that we will need to use the same instructions as the ones used in the Form1_Load event handler to display the first record.

So, in order to not duplicate code (something to avoid in any language, at any price), we think it would be a good idea to build a method which we could call and reuse for displaying a record.

In order to do so, highlight the lines you’d like to remove from the Form1_Load event handler and add to the new method, then right click on the selected code and choose Refactor / Extract Method...

You are then prompted to enter a method name: type: DisplayRecord:
then click OK.

Visual Studio automatically refactors your code to the following, automatically for you:

```csharp
private void Form1_Load(object sender, EventArgs e)
{
    cbDept.DataSource = (string[])apl.CallFn("GetDepts");
    DisplayRecord();
}

private void DisplayRecord()
{
    object[] record1 = (object[])apl.CallFn("GetRecord", 1);
    cbDept.SelectedIndex = (int)record1[0] - 1;  //department
    txtName.Text = (string)record1[1];           //name
    txtFirstName.Text = (string)record1[2];      //first name
    dtpBirthDate.Value = new DateTime(           //birth date
        ((int[])record1[3])[0],
        ((int[])record1[3])[1],
        ((int[])record1[3])[2])
};
```

This is almost perfect. However, we would like to pass the record number as an argument to the `DisplayRecord` method.

We can easily do this as follows (changes highlighted):

```csharp
private void Form1_Load(object sender, EventArgs e)
{
    cbDept.DataSource = (string[])apl.CallFn("GetDepts");
    DisplayRecord(1);
}

private void DisplayRecord(int recno)
{
    object[] record1 = (object[])apl.CallFn("GetRecord", recno);
    cbDept.SelectedIndex = (int)record1[0] - 1;  //department
    txtName.Text = (string)record1[1];           //name
    txtFirstName.Text = (string)record1[2];      //first name
    dtpBirthDate.Value = new DateTime(           //birth date
        ((int[])record1[3])[0],
        ((int[])record1[3])[1],
        ((int[])record1[3])[2])
};
```
Our next step is to enable the BindingNavigator buttons.

29. Create a new method called EnableButtons as follows:

```csharp
private void EnableButtons()
{
    bindingNavigatorMoveFirstItem.Enabled = true;
    bindingNavigatorMoveLastItem.Enabled = true;
    bindingNavigatorMoveNextItem.Enabled = true;
    bindingNavigatorMovePreviousItem.Enabled = true;
}
```

30. Add the following statements at the top of the Form1 class, just below the apl declaration:

```csharp
private int recordNo;
private int totalRecs;
```

This declares a “global variable” recordNo as an integer. This variable will hold the currently displayed record number. The totalRecs variable will hold the total number of records in our APL database.

31. Add the following statement at the beginning of the DisplayRecord method to update the recordNo variable:

```csharp
recordNo = recno;
```

32. Add the following statement to the Form1_Load event handler:

```csharp
totalRecs = (int)apl.CallFn("GetNumberOfRecords");
```

This initializes the totalRecs variable once for all (remember it is a “global variable”) to the total number of records in the APL database.

33. Double click on the bindingNavigatorMoveNextItem button in the BindingNavigator control to create its Click event handler and add the following code:

```csharp
private void bindingNavigatorMoveNextItem_Click(object sender, EventArgs e)
{
    DisplayRecord(recordNo + 1);
}
```

34. Double click each of the 3 other BindingNavigator buttons and enter the following code:

```csharp
private void bindingNavigatorMovePreviousItem_Click(object sender, EventArgs e)
{
    DisplayRecord(recordNo - 1);
}

private void bindingNavigatorMoveLastItem_Click(object sender, EventArgs e)
{
    DisplayRecord(totalRecs);
}
private void bindingNavigatorMoveFirstItem_Click(object sender, EventArgs e)
{
    DisplayRecord(1);
}

We must now get sure the BindingNavigator control displays the current record and the total number of records. The right place to put this code in is the DisplayRecord method of course.

35. Add the following 2 instructions at the end of the DisplayRecord method:

```csharp
bindingNavigatorCountItem.Text = "of " + totalRecs.ToString();
bindingNavigatorPositionItem.Text = recordNo.ToString();
```

These 2 instructions set the Text property of the BindingNavigator controls used to display the current record and the total number of records. Note that since a Text property is of type string, we must use the ToString() method to convert the recordNo and totalRecs integer variables to strings.

36. Now you can test the application. Just click on the Visual Studio toolbar button to start the application.

The form should display the first record from our APL Colossal File database:

![Image of the APL Colossal File database form displaying the first record]

and you can click on the BindingNavigator buttons to display the various database records:

![Image of the APL Colossal File database form with different records]

You may note that the birth date is displayed using the current culture of the local computer running the application. This works without us having had to write any line of code!
Note that each time you click on a button, a call to the GetRecord APL function is made, the APL Colossal File database is opened, a record is read from this database and returned to the C# application, then displayed in the form.

As you can see by testing this application, all this is absolutely instantaneous.

We have one more step to perform to finish our APL database records navigation system: when we reach either end of the database, we should disable the corresponding buttons to prevent the user from trying to go past the end of the database.

37. Change the EnableButtons method as follows (changes highlighted):

```csharp
private void EnableButtons()
{
    bindingNavigatorMoveFirstItem.Enabled = (recordNo > 1);
    bindingNavigatorMoveLastItem.Enabled = (recordNo < totalRecs);
    bindingNavigatorMoveNextItem.Enabled = (recordNo < totalRecs);
    bindingNavigatorMovePreviousItem.Enabled = (recordNo > 1);
    bindingNavigatorCountItem.Enabled = true;
}
```

Note that we must also call the EnableButtons method each time we use the DisplayRecord method, so add a call to EnableButtons at the end of the DisplayRecord method:

```csharp
private void DisplayRecord(int recno)
{
    recordNo = recno;
    object[] record1 = (object[])apl.CallFn("GetRecord", recno);
    cbDept.SelectedIndex = (int)record1[0] - 1;  //department
    txtName.Text = (string)record1[1];           //name
    txtFirstName.Text = (string)record1[2];      //first name
    dtpBirthDate.Value = new DateTime(           //birth date
        ((int[])record1[3])[0],
        ((int[])record1[3])[1],
        ((int[])record1[3])[2]);
    bindingNavigatorCountItem.Text = "of " + totalRecs.ToString();
    bindingNavigatorPositionItem.Text = recordNo.ToString();
    EnableButtons();
}
```

Our application is now finished as far as displaying records from the database is concerned. But we would like to add one more functionality: the ability to add new records to the database.

38. To do so, add a button to the BindingNavigator control.

Note: increase the form width if there’s not enough room to display the button
Change its Name property to: bnSave
Change its DisplayStyle property to: Text
Change its Text property to: Save

39. Add the following statement to the EnableButtons method:

```csharp
bindingNavigatorAddNewItem.Enabled = true;
```
40. Double click on the + BindingNavigator button to create its Click handler and enter the following code:

```csharp
private void bindingNavigatorAddNewItem_Click(object sender, EventArgs e)
{
    cbDept.Text = "";
    txtName.Text = "";
    txtFirstName.Text = "";
    dtpBirthDate.Value = new DateTime(1900,1,1);
    bindingNavigatorMoveFirstItem.Enabled = false;
    bindingNavigatorMoveLastItem.Enabled = false;
    bindingNavigatorMoveNextItem.Enabled = false;
    bindingNavigatorMovePreviousItem.Enabled = false;
    bindingNavigatorCountItem.Enabled = false;
    bindingNavigatorCountItem.Text = "";
    bindingNavigatorPositionItem.Text = "";
    bnSave.Enabled = true;
}
```

This code is so simple to understand that I won’t explain it. Simply note that it is not obvious to empty the content of the dtpBirthDate control, so I am displaying a default birth date of 1 January 1900. In a real application I would use the 2 instructions that allow to empty this control (which consists in setting the Format property to Custom and the CustomFormat property to an empty string). 

41. Now double click on the BindingNavigator bnSave button and enter the following code, which is a little bit more complex:

```csharp
private void bnSave_Click(object sender, EventArgs e)
{
    if (cbDept.Text.Trim().Length == 0
        || txtName.Text.Trim().Length == 0
        || txtFirstName.Text.Trim().Length == 0
        || dtpBirthDate.Value == new DateTime(1900, 1, 1))
        MessageBox.Show(
            "Please enter valid data in each field",
            this.Text,
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop
        );
    else
    {
        int dept = cbDept.SelectedIndex + 1;
        string name = txtName.Text.Trim();
        string firstName = txtFirstName.Text.Trim();
        int[] birthdate = new int[]{
            dtpBirthDate.Value.Year,
            dtpBirthDate.Value.Month,
            dtpBirthDate.Value.Day
        };
        object[] arg = { dept, name, firstName, birthdate };
        object result = apl.CallFn("AddRecord", arg);
        if (result is string)
            MessageBox.Show(
                (string)result,
                this.Text,
                MessageBoxButtons.OK,
                MessageBoxIcon.Error
            );
    }
```
Let's comment this code:

The initial if statement:

```csharp
if (cbDept.Text.Trim().Length == 0
    || txtName.Text.Trim().Length == 0
    || txtFirstName.Text.Trim().Length == 0
    || dtpBirthDate.Value == new DateTime(1900, 1, 1))
```

checks that none of the fields is empty and that the Birth Date field has been changed from the default: 1 January 1900.

| | is the “or” primitive in C#.

Note how you can call methods and properties in cascade, for example:

```csharp
cbDept.Text.Trim().Length
```

In this expression, `cbDept.Text` extracts the content of the `cbDept` ComboBox, then the `Trim()` method is applied to remove leading and trailing blanks, then the `Length` property is applied to calculate the resulting string length.

In case, the if statement returns true, we display a MessageBox with an error message.

```csharp
MessageBox.Show(
    "Please enter valid data in each field",
    this.Text,
    MessageBoxButtons.OK,
    MessageBoxIcon.Stop
);
```

Otherwise, the program runs the else part of the code.

This is the interesting part that will call the APL+Win AddRecord function to add a new record to our Colossal File database.

We first extract data from the form controls and store them in 4 local variables (dept, name, firstname and birthdate):

```csharp
int dept = cbDept.SelectedIndex + 1;
string name = txtName.Text.Trim();
string firstname = txtFirstName.Text.Trim();
int[] birthdate = new int[]{
    dtpBirthDate.Value.Year,
    dtpBirthDate.Value.Month,
}
dtpBirthDate.Value.Day
};

We then create an array of objects called `arg`, out of these 4 variables as follows:

```csharp
object[] arg = { dept, name, firstname, birthdate };
```

This is necessary because we need to pass a nested vector to the `AddRecord` APL function.

The next instruction calls the `AddRecord` APL+Win function, passing the `arg` variable as its argument:

```csharp
object result = apl.CallFn("AddRecord", arg);
```
Here is how the **AddRecord** APL function is defined:

```apl
R+AddRecord data;tie
[1] R+AddRecord data -- Add a new record to the database
[2] and returns its record number
[3]
[4] :if 4=≠data
[5] :andif 3≠4=data
[6] :andif 3≠4=data
[8] R+(data Dcfappend tie=TieDb)-DbOffset
[9] :else
[10] R+''Not saved: wrong data!''
```

```apl
R+DecodeDate date
[1] R+DecodeDate date -- Transforms a 3-element vector (yyyy mm dd)
[2] date into a scalar (yyymmdd) date
[3] Note: Belongs to the Business Layer
[4]
[5] R+100<date
```

Note that we have isolated the little piece of **Business Logic** into a separate **DecodeDate** function. The **AddRecord** function is part of the **Data Layer** and the **DecodeDate** function is part of the **Business Layer**.

**Why is this so important?**

Imagine that we one day we decide to change the **BirthDate** format in the database (currently **BirthDates** are saved as **YYYYMMDD** integers in the database). In such a case, having totally separated **DecodeDate** from **AddRecord**, we would only have to change **DecodeDate** and we would not have to make any change at all to our Data Layer (i.e. to the **AddRecord** function).

Note that we capture the result of the **AddRecord** function as an **object** (called **result**) in C#: the **AddRecord** APL function may either return an integer representing the record number just added to the database or a character string error message.

We must therefore check if the **result** variable is a **string** and in this case, display a **MessageBox** error message:

```csharp
if (result is string)
    MessageBox.Show(
        (string)result,
        this.Text,
        MessageBoxButtons.OK,
        MessageBoxIcon.Error
    );
```

If **result** is not a **string**, we know it is an **integer** and we know the record has been successfully added to the database, so we can display it in our User Interface:

```csharp
else
{
    totalRecs = (int)result;
    DisplayRecord(totalRecs);
    bnSave.Enabled = false;
}
```
Our application is now done. We can test it.

42. Click the toolbar button to test the application.

Click the BindingNavigator button to start entering a new record.

When done, click the Save button to save the record to the APL+Win Colossal File database: the form then display the record with its record number and total number of records updated; the navigation buttons are enabled:

**Conclusion**

In this section we have developed a simple albeit complete Client+Server C#+APL+Win application.

We have seen how C# handles the whole User Interface and APL handles the database and calculations, thus forcing a perfect separation between the User Interface Layer and the Data and Business Layers.

We’ve seen how easy it is to call APL functions from C# and pass them arguments which maybe nested arrays (of any depth and complexity) and how easy it is to get results sent back from APL again as nested arrays if needed.

We have also seen how to handle errors that may occur in APL and how to display them in the C# User Interface.
As a summary here is the complete code of the APL CONF2008.W3 workspace:

```
R AddRecord data;tie
   © AddRecord data -- Add a new record to the database
   © and returns its record number
   if 4=½data
      andif 323 82 82 323=Dr¨data
      andif 3=½4œdata
      R (data cfappend tie=TieDb)-DbOffset
   else
      R 'Not saved: wrong data!
   endif

BuildDb;dbName;tie;depts;Z;persons
   © BuildDb -- Builds a small Sample Human Resources Enterprise Database
   dbName='DbName
   © Create database
   tie=1+—/Œcfnums,0
   if FileExist dbName
      dbName œcftie tie
      dbName œcferase tie
   endif
   dbName œcfcreate tie
   © Create data
   depts=’Engineering ’’Human Resources’ ’Marketing’ ’Sales’ ’Production’
   persons=10 4½›’’
   persons[;1]=?10œdepts
   ’Groutsch’ ’Tillman’
   persons[;4]=19500100+¼10
   © Populate database
   Z=depts cfappend tie
   Z=’(’œcfappend¨9½tie
   Z=(Osplit persons) cfappend¨tie

R DbName
   © DbName -- Returns the database full path name
   R 'c:\aplwin\ele\conf2008.sf'

R DbOffset
   © DbOffset -- Returns number of component preceding first record in database
   R=10

R DecodeDate date
   © DecodeDate date -- Transforms a 3-element vector (yyyy mm dd) date
   © into a scalar (yyyymmdd) date
   © Note: Belongs to the Business Layer
   R=100ƒ¹date

R EncodeDate date
   © EncodeDate date -- Transforms a scalar (yyyymmdd) date
   © into a 3-element vector (yyyy mm dd)
   © Note: Belongs to the Business Layer
   R=œ10000 100 100 date
```
```apl
' exists•FileExist filename:h:max_path
  ¬ exists•FileExist filename -- Determine if a file exists
  max_path•0call 'W_Const' 'MAX_PATH'
  h•0call 'FindFirstFile'filename((11×4)+max_path+14)0Dtcnu1
  :if exists=h0call 'W_Const' 'INVALID_HANDLE_VALUE'
  :endif

' R•GetDepts;tie
  ¬ R•GetDepts -- Returns the list of all departments
  tie=TieDb
  R•Cfread tie,1

' R•GetNumberOfRecords;tie
  ¬ R•GetNumberOfRecords -- Returns the total number of records in the database
  tie=TieDb
  R•'1+2×Cfsize tie

' R•GetRecord number;tie;offset
  ¬ R•GetRecord number -- Retrieves record <number> from database
  tie=TieDb
  :if number>0
  :andif number<(2×Cfsize tie)–DbOffset
  R•Cfread tie,number+DbOffset
  R•'EncodeDate R•' business layer EncodeDate function
  :else
  R•'No record at this location!'
  :endif

' R•TieDb
  ¬ R•TieDb -- Ties the database and returns tie number
  R•DbName Cfstie 0
```
Here is the complete code of the C# Form1.cs class:

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using LescasseConsulting.AplServer;

namespace APL2000.Conf2008App
{
    public partial class Form1 : Form
    {
        private AplServer apl;
        private int recordNo;
        private int totalRecs;

        public Form1()
        {
            InitializeComponent();
            apl = new AplServer(@"C:\aplwin\ele\conf2008");
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            bindingNavigator1.Enabled = true;
            cbDept.DataSource = (string[])apl.CallFn("GetDepts");
            totalRecs = (int)apl.CallFn("GetNumberOfRecords");
            DisplayRecord(1);
            EnableButtons();
        }

        private void DisplayRecord(int recno)
        {
            recordNo = recno;
            object[] record1 = (object[])apl.CallFn("GetRecord", recno);
            cbDept.SelectedIndex = (int)record1[0] - 1;  //department
            txtName.Text = (string)record1[1];           //name
            txtFirstName.Text = (string)record1[2];      //first name
            dtpBirthDate.Value = new DateTime(           //birth date
                ((int[])record1[3])[0],
                ((int[])record1[3])[1],
                ((int[])record1[3])[2]);
            bindingNavigatorCountItem.Text = "of " + totalRecs.ToString();
            bindingNavigatorPositionItem.Text = recordNo.ToString();
            EnableButtons();
        }

        private void EnableButtons()
        {
            bindingNavigatorMoveFirstItem.Enabled = (recordNo > 1);
            bindingNavigatorMoveLastItem.Enabled = (recordNo < totalRecs);
            bindingNavigatorMoveNextItem.Enabled = (recordNo < totalRecs);
            bindingNavigatorMovePreviousItem.Enabled = (recordNo > 1);
            bindingNavigatorCountItem.Enabled = true;
            bindingNavigatorAddNewItem.Enabled = true;
        }

        private void bindingNavigatorMoveNextItem_Click(object sender, EventArgs e)
        {
            DisplayRecord(recordNo + 1);
        }
    }
}
```
private void bindingNavigatorMovePreviousItem_Click(object sender, EventArgs e)
{
    DisplayRecord(recordNo - 1);
}

private void bindingNavigatorMoveFirstItem_Click(object sender, EventArgs e)
{
    DisplayRecord(1);
}

private void bindingNavigatorAddNewItem_Click(object sender, EventArgs e)
{
    cbDept.Text = "";
    txtName.Text = "";
    txtFirstName.Text = "";
    dtpBirthDate.Value = new DateTime(1900, 1, 1);
    bindingNavigatorMoveFirstItem.Enabled = false;
    bindingNavigatorMoveLastItem.Enabled = false;
    bindingNavigatorMoveNextItem.Enabled = false;
    bindingNavigatorMovePreviousItem.Enabled = false;
    bindingNavigatorCountItem.Enabled = false;
    bindingNavigatorCountItem.Text = "";
    bindingNavigatorPositionItem.Text = "";
    bnSave.Enabled = true;
}

private void bnSave_Click(object sender, EventArgs e)
{
    if (cbDept.Text.Trim().Length == 0
        || txtName.Text.Trim().Length == 0
        || txtFirstName.Text.Trim().Length == 0
        || dtpBirthDate.Value == new DateTime(1900, 1, 1))
    
        MessageBox.Show(
            "Please enter valid data in each field",
            this.Text,
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop
        );

    else
    {
        int dept = cbDept.SelectedIndex + 1;
        string name = txtName.Text.Trim();
        string firstname = txtFirstName.Text.Trim();
        int[] birthdate = new int[]
            { dtpBirthDate.Value.Year,
            dtpBirthDate.Value.Month,
            dtpBirthDate.Value.Day
            };
        object[] arg = { dept, name, firstname, birthdate };
        object result = apl.CallFn("AddRecord", arg);
        if (result is string)
            MessageBox.Show(
                (string)result,
                this.Text,
                MessageBoxButtons.OK,
                MessageBoxIcon.Error
            );
        else
        {
            totalRecs = (int)result;
            DisplayRecord(totalRecs);
            bnSave.Enabled = false;
        }
    }
}
Developing a Client Server ClickOnce C# + APL+Win Application

Introduction

Now that we have seen how to write a Client Server ClickOnce application using C# and APL+Win, we can push the idea one step further.

Wouldn’t it be nice if the Server were a Remote Computer?

And wouldn’t it be really nice if we could launch such an application from Internet?

The answer is: all this is possible and not very difficult.

Explaining how to do this in detail is out of the scope of this presentation, but I will at least expose the principles and show a real life example.

Principles

Here are the 10 major steps needed to create a Client-Server ClickOnce C#+APL+Win application:

1. You must have an Internet Remote Server, running Windows Server 2003 or Windows Server 2008
2. You must install APL+Win v6.4+ on the Remote Server
3. You must also have APL+Win v6.4+ installed on your development computer
4. You must have Visual Studio 2005 or better Visual Studio 2008 installed on your development computer (but not on the Server)
5. You must create your application using C# and APL+Win, just as we did in the previous chapter, though you should use the LescasseConsulting.AplRemoting DLL which is similar to LescasseConsulting.AplServer, but slightly more complex
6. You should use the .Net Remoting functionnalities to register a Channel and register and activate an instance of the LescasseConsulting.AplRemoting object in your application C# main form
7. You should create another Visual Studio application, choosing the Windows Service template to create a Windows Service which will be installed on the Server: the role of this Service will be to open a channel for the incoming requests and transmit them to the Server application (APL).
8. You need to install the Windows Service on the development computer and on the Server
9. You should allow the Server to use a port (other than 80) for use by this Client-Server application
10. You must publish your application as a ClickOnce application using the Properties / Publish tab

There are other details to take care of, but basically, once you have understood how to do all this, it is not very complex.

With a little habit, you can build a basic Client-Server ClickOnce C#+APL+Win application in half a day.

Development Process

One important aspect of the development of a Client-Server ClickOnce C#+APL+Win application is that you use your development computer as the Server during the whole application development and test phase.
This is pretty easy to do by simply setting the application publish URL to:

http://localhost/app_name

instead of setting it to the final destination of your application which will be on your Server, such as:

http://www.lescasse.com/app_name

This allows you to develop and test your application locally, using your computer as both the Client machine and the Server.

**Debugging a ClickOnce application**

Debugging a ClickOnce application is most easy under Windows XP\(^2\), because the ActiveX APL Session shows up as you start the application.

Therefore, at any time, you can use:

1. Either Visual Studio 200x to fix any problem that might arise on the C# side of your application
2. Or APL+Win to fix any problem that might arise on the APL side of the application

Debugging is really easy.

If you get an error in APL, the C# application will temporarily freeze: you can just hit Alt+Tab to go to the APL Session, fix the problem and branch to quadLC to resume it! It is as easy as that.

And of course you can put stops or traces in your APL programs (the ones called by C#) and check the arguments sent by C#, check the result sent back to C#, etc.

Similarly, you can use all the excellent Visual Studio standard debugging features to debug your C# application (Intellisense, stops, stepping thru your code instruction by instruction, watch window, immediate execution window, etc.)

**Example**

Here is a really complex and large APL application which I have ported to Internet as a Client-Server ClickOnce C#+APL+Win application for one of my customers. I am presenting it here with his permission. I am only going to show a couple of Dialog Boxes from this application as an example.

This APL application had required years to be developed, had been in production for a number of years and is constantly evolving. It had become a fairly large one with hundreds of APL functions, some way over 1000 lines of really complex APL code.

Porting this application to Internet has required some time and efforts, but the whole port has taken less than 60 man days.

\(^2\) Under Windows Vista debugging is a little more tedious because Vista isolates the C# application and the APL application and you need to click on a button and wait a couple of seconds to shift from one to the other one.
The result is an Internet ClickOnce Windows application which does not require any installation on client computers and has many advantages over the previous 100% APL version of the application.

For example, when my customer needs to have his own many customers use a new version of the application, he no longer needs to send all of them a CD and have them install the update or ask them to download and install a new version locally on their computer. This used to require a lot of time and efforts. Now, he can simply upload the new version to his Internet Server and boom! all his customers are instantly using the new version without any installation required.

I will show you the application on my local computer, but it would be exactly the same if I were using it from Internet, except for the URL I’d be using.

To launch the application, I start Internet Explorer and enter the http://localhost/Etools/publish.htm URL:

![Image of the ClickOnce page](image)

The above page is typical of a ClickOnce application: it is the standard page that’s produced by Visual Studio when you publish the application. It includes the Company and Application Name, the Application Version, the Publisher and the Run button to start the application.

Note that it is pretty easy to customize this page of course, to better integrate it within the customer’s Web Site, using his Web Site’s look and feel.

Let’s click on the Run button: 2 things happen:

- An ActiveX APL Session pops up with the application workspace loaded
- A C# Login dialog box gets displayed
You must enter your **Login** and **Password** in the C# Login form: if your password is older than 30 days, you are forced to change it and enter a new one (see above screenshot).

Note also that the **Login** and **Password** are encrypted (using one of the many .NET Framework official encryption classes) so that they do not travel in clear on Internet.

The **ETools Login** form is already part of the Client-Server C#+APL+Win process, i.e. the **ETools Login** dialog is a pure C# form (since C# handles 100% of the User Interface), but as you hit **Tab** after entering your **Login**, calls to APL are made to compare the Login you entered against the registered users in the ETools APL Users database.
After you click Login, the application starts.

The ETools application is an MDI C# application.

Click on Admissions / Admissions Reports...

A dialog box is displayed:

If you click on the ... button to the right of the Group(s) TextBox, the following form gets displayed: its Groups and Variables list box is populated with data coming from calls made to APL functions which retrieve data from the ETools database.
Let's select **Admit** and click **OK**.

Complete the **Group(s)** TextBox and fill the **Build Report / Figure** as follows then click the **Show Report** button:

As you click **Show Report** a request is made to APL on the Server to extract the right data from the database and compute a report which is returned to C# as a set of nested arrays. C# then uses these nested arrays to produce and display a report (see screenshot above). Note that the APL functions used to compute this report are complex and that hundreds of lines of APL code are executed in the background.

Click on the **Build Figure** check box: this enlarges the **Build Report / Figure** dialog and displays various settings which we can change to produce a customized graph.

Note that the buttons caption have changed (for example: **Show Report** has become **Show Figure**). Click on the **Show Figure** button.
Change a few settings in the **Figure Properties** frame: you don’t need to click the **Show Figure** button again since any change to a setting is instantly reflected on the graph.
The graphs used in this application are produced by the Adrian Smith’s SharpPlot product.

You may ask me: what do the graphs have to do with APL, if they are produced by a C# DLL?

Well, nothing really, I must admit, but a lot of other stuff in this Build Report / Figure dialog requires calls to functions in the APL workspace.

Conclusion

This example shows how you can really mix C# (to build your User Interface, possibly also using third party products like the SharpPlot DLL) and use APL at the same time in the background to run all the existing Business Logic of your APL application and to access your APL databases (whether component files, colossal files, text files, etc.)

\[23\] SharpPlot is an excellent .Net product (as usual with everything Adrian Smith produces) which you can easily use in any C# application
Developing an ASP.Net Web Site using APL+Win in the Background

Introduction

The last aspect I would like to talk about is the possibility to develop pure Web Sites using APL+Win in the background.

If you also use a bit of Ajax on top of it, your Web Pages will get updated by APL without any flicker (i.e. without any visible reloading of the page).

Examples

Let’s show a couple of examples\textsuperscript{24} that I have built for this paper.

Let’s start Visual Studio 2008 and open a WebSite project I called LcNew.

Let’s start this Web Site by clicking the toolbar button.

First thing you’ll notice is that an instance of the APL+Win ActiveX Server pops up with the TESTWEB workspace loaded.

\textsuperscript{24} You can visit these examples at: http://www.lescasse.com/lcnew
Then, an instance of Internet Explorer pops up with the **Web Site Home Page** displayed:

![Web Site Home Page](image.png)

But, let’s click on the **RandomTable** link in the left menu.
This loads the RandomTable.aspx page:

![RandomTable.aspx screenshot]

This page displays a form looking like a Windows XP form\(^\text{25}\).

Every time you click the Refresh button, the following APL function is called in the background and the result is returned to the C# code-behind page and then displayed in the Web page:

```apl
∇ R←RandomTable rargs;rows;cols
[1]   (rows cols)←rargs
∇
```

Please note that the Web Page is almost instantly refreshed with new random data and that this happens without the page being apparently reloaded!

You may change the number of Rows and/or Columns and click the Refresh button. This of course calls our RandomTable APL function and dynamically extends the result table.

---

\(^{25}\) But this is using some Web Site graphics to make it look like a Windows XP form.
Here is a slightly more complex example inspired from a similar Dyalog APL example.

Click on the Loan link in the left menu. The following Web page gets displayed:

The results displayed in the Web Page HTML table are calculated by a CalcPayments APL function.

Each time you change a setting in the Loan Example form, the CalcPayments APL function is rerun in the background and the results displayed in the Web Page HTML table object.
Note again, that this happens without the Web Page being apparently reloaded and with no flicker at all.

You may click on the Explanations blue bar (an Ajax Control Toolkit CollapsiblePanelExtender object) to display some explanations and code samples showing how this page works:

```csharp
private void CalcPayments()
{
    // Read the form fields
    double loanAmnt = 0;
    int max = 0, min = 0;
    double max = 0, min = 0;
    Double.TryParse(LoanAmount.Text, out loanAmnt);
    Int32.TryParse(LenMax.Text, out max);
    Int32.TryParse(LenMin.Text, out min);
    Double.TryParse(IntrMax.Text, out max);
    Double.TryParse(IntrMin.Text, out min);
    int perType = 1 + (PeriodType.Checked / 1 == 1);

    // Set the APL+Win PeriodType variable
    apICalls.APLGetVariable("PeriodType", perType);

    // Call the APL+Win CalcPayments function
    object[] rarg = { loanAmnt, max, min, max, min, max, min };
    double[] payments = Transpose(double[](double[] apICalls.APLCallFn("CalcPayments", rarg)));

    // Populate the ASP.Net Gridview object
    grid1.DataSource = ToDataTable(payments);
    grid1.DataBind();
    grid1.Columns[1].DefaultCellStyle.Width = 80;
}
```

Another aspect of this Web Site application is that the APL ActiveX Session gets unloaded from memory after one minute of inactivity. However, as soon as you use the Load Payments form again, the APL+Win ActiveX Session gets automatically loaded and the CalcPayments function is used. This mechanism which I am not
going to show or explain here, allows to spare Server resources, especially when your Web age is used by many visitors.
ASP.Net Code

Although the following might look too complex to many, I am still presenting it for those who already have some C# and APL.Net experience and might be interested.

Here is the ASP.Net code for this Web Page:

```xml
<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="ajaxToolkit" %>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentTitle" Runat="Server">
Loan Example <span class="subtitle">(calling APL+Win from an ASP.Net Web page)</span>
</asp:Content>

<asp:Content ID="Content4" ContentPlaceHolderID="LastUpdate" Runat="Server">
Last Updated: 23 mar 2008
</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">

<p>This example shows an <b>ASP.Net Web Form</b> which uses <b>APL+Win</b> in the background to calculate a Loan.</p>

<p>Enter data for your Loan in the following Form: as you do so, the yearly or monthly payments are calculated by <b>APL+Win</b> in the background and immediately displayed in a grid below the form.</p>

<table cellspacing="0" cellpadding="0" class="formtable">
<tr>
<td colspan="2" class="formcaption">
<table width="100%">
<tr>
<td align="left" width="20"><img src="images/captionicon.gif" alt=""></td>
<td align="left" style="font: bold 110% Verdana, Arial, sans-serif; color:White;">Loan Example</td>
<td align="right"><img src="images/captionclose.gif" alt=""></td>
</tr>
</table>
</td>
</tr>
<tr>
<td class="formcell">
<asp:Label ID="lLoanAmt" runat="server" Text="Purchase Amount" CssClass="aspLabel" Height="14px"></asp:Label>
<asp:TextBox ID="LoanAmt" runat="server" CssClass="aspTextBox" ontextchanged="LoanAmt_TextChanged" AutoPostBack="True"></asp:TextBox>
</td>
<td class="formcell">
<asp:CheckBox ID="PeriodType" runat="server" Text="Period are years" CssClass="aspCheckBox" oncheckedchanged="PeriodType_CheckedChanged" AutoPostBack="True"></asp:CheckBox>
</td>
</tr>
</table>
</asp:Content>
```
As you make changes to the above Web Form, events are triggered in the C# code-behind page. Here is an example:

```csharp
protected void PeriodType_CheckedChanged(object sender, EventArgs e)
{
    CalcPayments();
}
```

All these events call the following C# CalcPayments method using the LescasseConsulting.AplRemoting DLL:

```csharp
private void CalcPayments()
{
    // Read the form fields
    double loanAmt = 0;
    int lmax = 0, lmin = 0;
    double imax = 0, imin = 0;
    Double.TryParse(LoanAmt.Text, out loanAmt);
    Int32.TryParse(LenMax.Text, out lmax);
    Int32.TryParse(LenMin.Text, out lmin);
    Double.TryParse(IntrMax.Text, out imax);
    Double.TryParse(IntrMin.Text, out imin);
    int pertype = 1 + (PeriodType.Checked ? 1 : 0);

    // Set the APL+Win PeriodType variable
    apiCalls.APLSetVariable("PeriodType", pertype);
}
```

The above C# CalcPayments method first reads the content of the Web Form fields, then sets the APL PeriodType variable and then calls the following APL+Win CalcPayments function:

```
'  PAYMENTS"CalcPayments
X;LoanAmt;LenMax;LenMin;IntrMax;IntrMin;PERIODS;INTEREST;NI;NM;PER;INT;Œelx
[1] © Calculates loan repayments
```
The results from the `<b>APL+Win CalcPayments</b>` function are displayed in an `<b>ASP.Net Gridview</b>` object which is dynamically populated.

Note that the input form at the top of this page has been made looking like a Windows XP Form, with the help of a little bit of `<b>CSS</b>` and a couple of images.

As you can see, it is possible to develop Web applications having a nice enough User Interface and using `<b>APL+Win</b>` in the background.

Note that `<b>Ajax</b>` is used on this page to prevent the entire page from reloading every time the resulting grid is calculated.
### Interface APL+Win and C#

```xml
<asp:Label>
</p>

<asp:GridView ID="GridView1" runat="server"
    EmptyDataText="No data!"
    ShowHeader="False"
    CssClass="aspGridView"
    onrowcreated="GridView1_RowCreated"
>
    <RowStyle CssClass="aspGridRowStyle" />
    <AlternatingRowStyle CssClass="aspGridAltRowStyle" />
</asp:GridView>

</asp:Content>
```
Here is the ASP.Net code behind page for this Web Page:

```csharp
using System;
using System.Collections;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using LescasseConsulting.AplRemoting;

public partial class Loan : System.Web.UI.Page
{
    public partial class Loan : System.Web.UI.Page
    {
        AplCalls aplCalls = (AplCalls)Session["aplCalls"]; 
        int[] lenmax, lenmin;
        double[] intrmax, intrmin;

        protected void Page_Load(object sender, EventArgs e)
        {
            ((HtmlTitle)Master.FindControl("PageTitle")).Text = "Loan Example";
            Label NbVisits = (Label)Master.FindControl("NbVisits");
            if (NbVisits != null)
            {
                NbVisits.Text = "Visits: " + Application["ActiveUsers"].ToString() + " on line";
            }
            aplCalls = (AplCalls)Session["aplCalls"]; 

            if (!IsPostBack)
            {
                lenmax = new int[40];
                for (int i = 0; i < lenmax.Length; i++)
                    lenmax[i] = i + 1;
                lenmin = new int[40];
                for (int i = 0; i < lenmin.Length; i++)
                    lenmin[i] = i + 1;
                intrmax = new double[40];
                for (double i = 0; i < intrmax.Length / 2; i += 0.5)
                    intrmax[(int)(2 * i)] = i + 0.5;
                intrmin = new double[40];
                for (double i = 0; i < intrmin.Length / 2; i += 0.5)
                    intrmin[(int)(2 * i)] = i + 0.5;
                LenMax.DataSource = lenmax;
                LenMin.DataSource = lenmin;
                IntrMax.DataSource = intrmax;
                IntrMin.DataSource = intrmin;
                Page.DataBind();
                LoanAmt.Text = "1000000";
                LenMax.Text = "10";
                LenMin.Text = "1";
                IntrMax.Text = "10";
                IntrMin.Text = "1";
                CalcPayments();
            }
        }
    }
}
```
protected void LenMax_SelectedIndexChanged(object sender, EventArgs e)
{
    CalcPayments();
}

protected void LenMin_SelectedIndexChanged(object sender, EventArgs e)
{
    CalcPayments();
}

protected void IntrMax_SelectedIndexChanged(object sender, EventArgs e)
{
    CalcPayments();
}

protected void IntrMin_SelectedIndexChanged(object sender, EventArgs e)
{
    CalcPayments();
}

protected void LoanAmt_TextChanged(object sender, EventArgs e)
{
    CalcPayments();
}

protected void PeriodType_CheckedChanged(object sender, EventArgs e)
{
    if (PeriodType.Checked)
        GridViewLabel.Text = "Yearly Payments by Interest Rates and Number of Years";
    else
        GridViewLabel.Text = "Monthly Payments by Interest Rates and Number of Years";
    CalcPayments();
}

private void CalcPayments()
{
    double loanAmt = 0;
    int lmax = 0, lmin = 0;
    double imax = 0, imin = 0;
    Double.TryParse(LoanAmt.Text, out loanAmt);
    Int32.TryParse(LenMax.Text, out lmax);
    Int32.TryParse(LenMin.Text, out lmin);
    Double.TryParse(IntrMax.Text, out imax);
    Double.TryParse(IntrMin.Text, out imin);
    int pertype = 1 + (PeriodType.Checked ? 1 : 0);
    aplCalls.APLSetVariable("PeriodType", pertype);
    object[] rarg = { loanAmt, lmax, lmin, imax, imin };
    double[,] payments =
        Transpose((double[,])aplCalls.APLCallFn("CalcPayments", rarg));
    GridView1.DataSource = ToDataTable(payments);
    GridView1.DataBind();
    for (int i = 0; i < GridView1.Columns.Count; i++)
        GridView1.Columns[i].ItemStyle.Width = 60;
}
public static T[,] Transpose<T>(T[,] matrix)
{
    T[,] result = new T[matrix.GetLength(1), matrix.GetLength(0)];
    for (int i = 0; i < result.GetLength(0); i++)
    for (int j = 0; j < result.GetLength(1); j++)
        result[i, j] = matrix[j, i];
    return result;
}

private static DataTable ToDataTable<T>(T[,] matrix)
{
    DataTable result = new DataTable();
    List<DataColumn> cols = new List<DataColumn>();
    for (int i = 0; i < matrix.GetLength(1); i++)
        cols.Add(new DataColumn());
    result.Columns.AddRange(cols.ToArray());
    T[] vec = new T[matrix.GetLength(1)];
    DataRow dr;
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        dr = result.NewRow();
        for (int j = 0; j < matrix.GetLength(1); j++)
            dr[j] = matrix[i, j];
        result.Rows.Add(dr);
    }
    return result;
}

protected void GridView1_RowCreated(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        e.Row.Height = 8;
        foreach (TableCell td in e.Row.Cells)
        {
            td.CssClass = "aspGridViewTD";
        }
    }
}
General Conclusion
As we have seen in this long paper, there are lots of ways with which you can interface APL+Win and C#.

These 2 languages, although very different, have a lot in common:

1. Both are very pure and clean
2. Both are very safe (they very rarely crash: C# and Visual Studio 2008 never crashed for me)
3. Both are a joy to use

But these 2 languages are also complementary:

1. C# and Visual Studio are close to perfect for building User Interfaces (it is so much easier and faster!)
2. APL is close to perfect for exploring your data, transforming them, calculating, even for storing your data (i.e. Colossal Files) sometimes

Finally these 2 languages interface really perfectly, with no overhead and no restrictions.

By interfacing C# and APL+Win you can:

1. Build real .Net applications which use APL+Win in the background: doing so, you can claim, and it is true, that your application has become a .Net application and surely enough reassure and satisfy your own customers
2. Build much nicer to look User Interfaces, more rapidly and more efficiently
3. Use the whole power of the .Net Framework!
4. Still use the APL+Win in the background for maybe the largest part of your application

My conclusion is that it is really worth making the effort to learn enough of C# to be able to interface C# and APL: you can only immensely benefit from doing so.

And if you are like me, you might well and soon discover that you love Visual Studio and C# as much as you love APL!

Finally, why not end this paper with a FAQ to answer some of the questions you may have in mind.
FAQ

Why is it important to interface APL+Win and C#?

Because you can then use the whole .Net Framework in your applications.
Because you can then write your application User Interfaces in C#, more easily than in APL.
Because you can write more attractive and faster User Interfaces this way.
Because you can port your application to Internet and make it a ClickOnce application.
Because there is more and more pressure everyday from customers to provide them with a .Net application.
Because you can still use APL+Win for a large part of your application, often for most of it.

What benefit can an APL developer get from interfacing APL+Win and C#?

By interfacing with C# you can easily solve problems which would be very difficult to solve in APL: for example, you can easily use Regular Expressions, exploit XML files using XSLT, use official encryption techniques, watch your file system, etc.

In general, you can also greatly improve your User Interfaces.

How difficult is it to interface APL+Win and C#?

It is very easy, and this document is a proof. However, this assumes that you learn enough of C# and of the .Net Framework. Once you have made these efforts, it is really easy.

How much should I learn about .Net and C#?

You should involve yourself in learning both C# and the .Net Framework. This certainly requires efforts. But I have done it and I consider that I am now about as fluent in C#/ .Net as I am in APL.

So, if I have done it, you can definitely do it. I can help, if you need so.

Where can I find help on learning .Net and C#?

There are hundreds of books on the C# language itself and about as many on the .Net Framework, plus books on Visual Studio, etc. Look at: http://www.lescasse.com/CSharpBooks.asp for a few.

There are hundreds of thousands of C# examples on Internet as well as many Web Sites including C# Tutorials, etc.
I have written a 500 pages C# course especially aimed at APL developers willing to interface APL and .Net C# and have already trained many people using this course.

I have recorded more than 20 hours of ScreenCasts teaching C# and showing practically how to develop real life C# applications. These are available on my Web Site. Look at: http://www.lescasse.com/CSharpTraining.asp

**In which ways can an APL developer interface to .Net C#?**

Here are a few ways (demonstrated in this paper) by which you can interface both languages:

1. write COM Interop C# DLLs that can then be directly used as ActiveX from APL+Win, using [.wi

2. write Client-Server applications where the User Interface is a .Net application written in C# and the business logic and calculations are written in APL

3. write the same kind of Client-Server application, but using a Remote Internet Server and transform your application into a ClickOnce Internet based application launchable from anywhere in the world, in a browser

4. write ASP.Net Web Sites using C# for the page layout and APL in the background for calculations

**What should the role of APL+Win and of C# be?**

If you write COM Interop DLLs in C#, the role of C# will be to give you access to the whole .Net Framework from APL.

If you write Client Server applications, C# will be the Client and you will want to write the whole application User Interface in C#, while APL+Win will be the Server and you’ll keep all the Business Logic (calculations, etc.) in APL as well as the Data Layer (for example, storing your data in APL Colossal Files).

**Why C# and not VB.Net?**

You could probably interface VB.Net with APL as well as C# with APL although I have not tried to explore this path in detail. However, I personally like the extreme rigor of C# and its purity which, for me, are qualities very close to similar qualities of the APL language. In general, I much prefer C# to VB.Net as a Programming Language: it is more declarative, but at the same time it is much more strict and clearer.
What IDE (Integrated Development Environment) should I use to develop my APL+C# application?

You could start with the free Microsoft Visual C# Express which is a “simpler” version of Visual Studio. Be sure to download and install the Visual C# Express 2008 version.

But you will pretty soon want to use the full featured Visual Studio 2008 Professional.

This development environment is just unbelievable. It is not a surprise that Microsoft has spent billions of dollars just on developing Visual Studio. And this sells for a very reasonable price.

It is also extremely secure: I am never experiencing any problem with Visual Studio 2008 (I used to rarely experience little problems with Visual Studio 2005, but the 2008 version is just quasi perfect).

How much does it cost?

There are several versions of Visual Studio 2008 available (Express, Standard, Professional and Team System). The version I recommend is Visual Studio 2008 Professional: the price is around 1000 USD. You can find a comparison chart at: http://msdn.microsoft.com/en-us/vstudio/aa700921.aspx

Should you use VS 2005 or VS 2008?

Do not hesitate one second: use Visual Studio 2008!

Is VS 2008 a secure development environment?

Yes, definitely.

I have a huge APL application: can I port it to C# + APL+Win?

Sure. Obviously, the larger the APL application, the larger the effort to port it to C# + APL+Win. But there are a number of techniques that could help in reducing the overall effort. For example, it would not be very difficult to write a translator application that would automatically convert APL dialog boxes to C# dialog boxes, at least as far as the User Interface is concerned.
Will my C#+APL+Win application be difficult to debug?

The answer is No!

It is as easy to debug a Client-Server C# + APL+Win application as it is to debug a pure APL application.

At any time, you have access to both the Visual Studio debugging facilities which are tremendous and to the APL+Win debugging facilities which are not bad either, as well as to the APL Session.

I would even go as far as saying that it is often easier to debug a C# + APL+Win application because, by nature, in such an application the User Interface is totally separated from the Business Logic and the Data Layer. This makes debugging easier and results in a much better architected application.

If .Net C# is so good why do I still need APL at all?

Good question: even though Visual Studio, C# and the .Net Framework are so good, it is still often easier and sometimes faster to manipulate data in APL, although Microsoft is progressively improving C# so much that this gap is getting constantly reduced. For example, the .Net Framework 3.5 includes a new technology called **Linq** which is very close to APL by many aspects and allows you to globally work on arrays (one dimensional arrays only for now) as you do in APL. **Linq** has operators which are often very similar to APL primitives (they even sometimes have the same name and behavior: example: **Take**)

The other and probably main reason why you still need APL is that you know APL very well and that your current application is written in APL. Therefore it is good to know that you can port it to .Net while still keeping large parts of it unchanged in APL.

Does interfacing C# and APL+Win really works in real life?

Yes.

I have ported more than 10 customers APL application, some fairly large, to .Net using technologies similar to the ones exposed in this document. The ETools application shown in this paper is an example.

Some of my customers have ported applications by themselves after having followed a C#+APL+Win Training, APL2000 and other customers as well as myself, have been successfully using NetAccess to develop C# DLLs and use them from APL+Win in production environments.

Interfacing APL+Win works and works really fine.