Porting your Dyalog Application to Internet

by
Eric Lescasse
Lescasse Consulting
18 rue de la Belle Feuille
92100 Boulogne
eric@lescasse.com
http://www.lescasse.com

September 11, 2007

Introduction

There are several approaches you can take to port your Dyalog application to Internet, but say you have the following goals of transforming your Dyalog application to:

- a Microsoft .Net application
- an application that could be used from a browser by anyone having an Internet connection
- an application that would remain a rich client application, i.e. a Windows application
- an application that would load and start on Client computers without installing any files on these computers
- an application that would still use your Dyalog functions/variables/files on the Server
- an application that can be launched with just one click by any user

This seems too good to be true, but this is indeed now possible.

Read on.

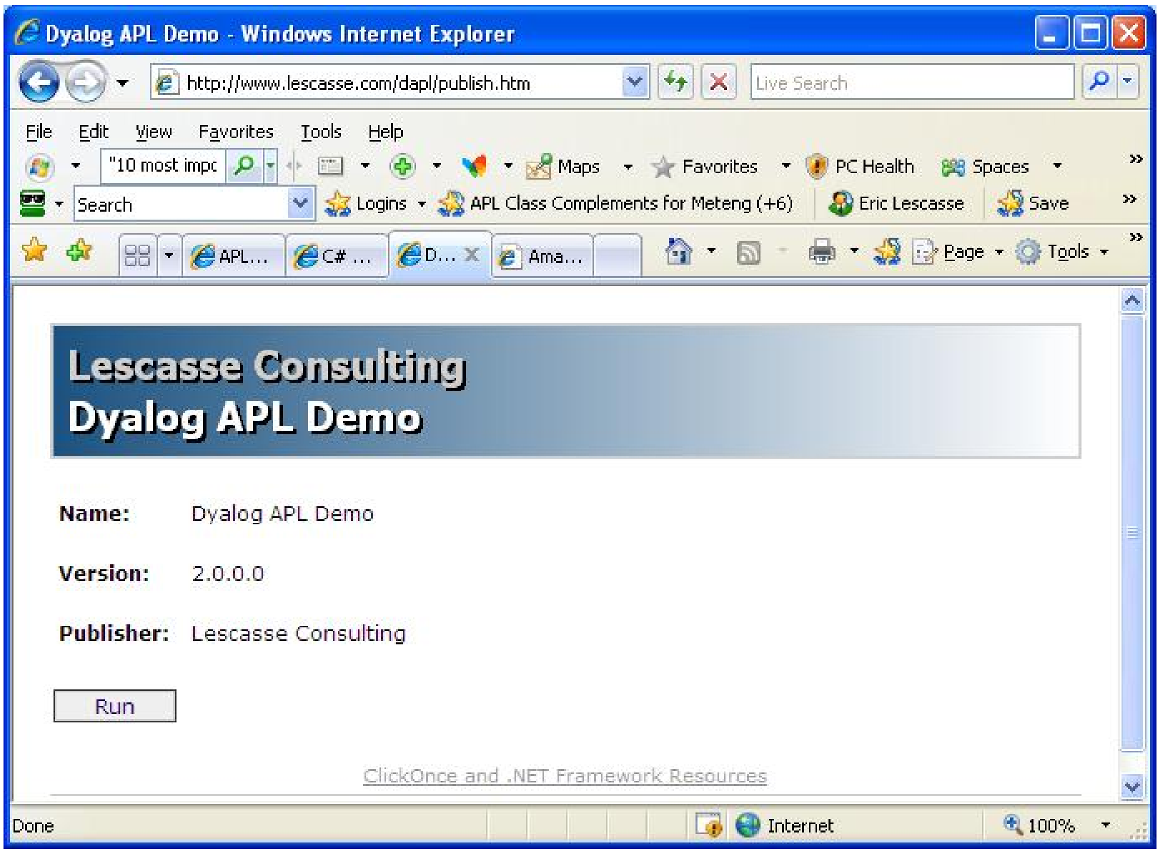
Sample Demo

First let us prove it is possible and let us show you see an example:

Open your Internet Explorer browser (hopefully version 6 or 7) and enter the following URL:

http://www.lescasse.com/dapl/publish.htm

You should be presented with the following Web page:


Dyalog APL Demo - Windows Internet Explorer
http://www.lescasse.com/dapl/publish.htm
Live Search
File Edit View Favorites Tools Help
Maps Favorites PC Health Spaces
Search Logins APL Class Complements for Meteng (+6) Eric Lescasse Save
APL... C# ... D... Ama...
Page Tools
Run
ClickOnce and .NET Framework Resources
Done Internet 100%

Lescasse Consulting
Dyalog APL Demo

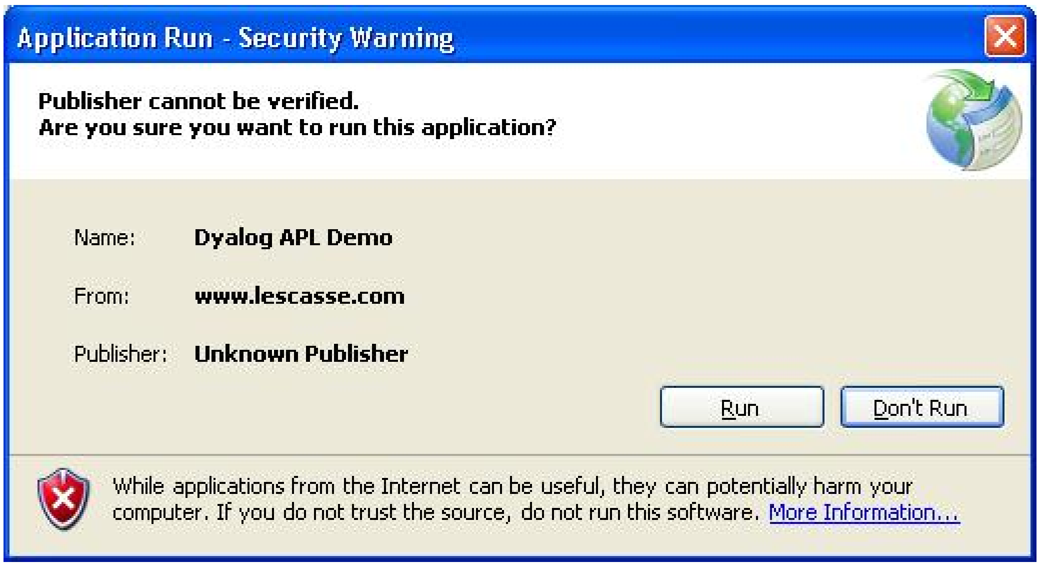
Name:	Dyalog APL Demo
Version:	2.0.0.0
Publisher:	Lescasse Consulting

Click on the **Run**[1] button.

A dialog gets displayed telling you that the application is loading.

Then another dialog prompts you to confirm you want to run this application[2].

[1] If your computer runs in an environment which is protected by a strong Firewall, you will have to authorize **port 10100** in the Firewall to be able to load this demo.
[2] It is possible to avoid having this additional dialog prompting the User for confirmation by getting a Trust Certificate. This is out of the scope of this paper. For those interested, there is a good MSDN article from Brain Noyes which can be found from http://tinysells.com/48 and which provides detailed insight into when prompting the user occurs as well as the ability to alter it.



Application Run - Security Warning

Publisher cannot be verified.
Are you sure you want to run this application?

Name: Dyalog APL Demo

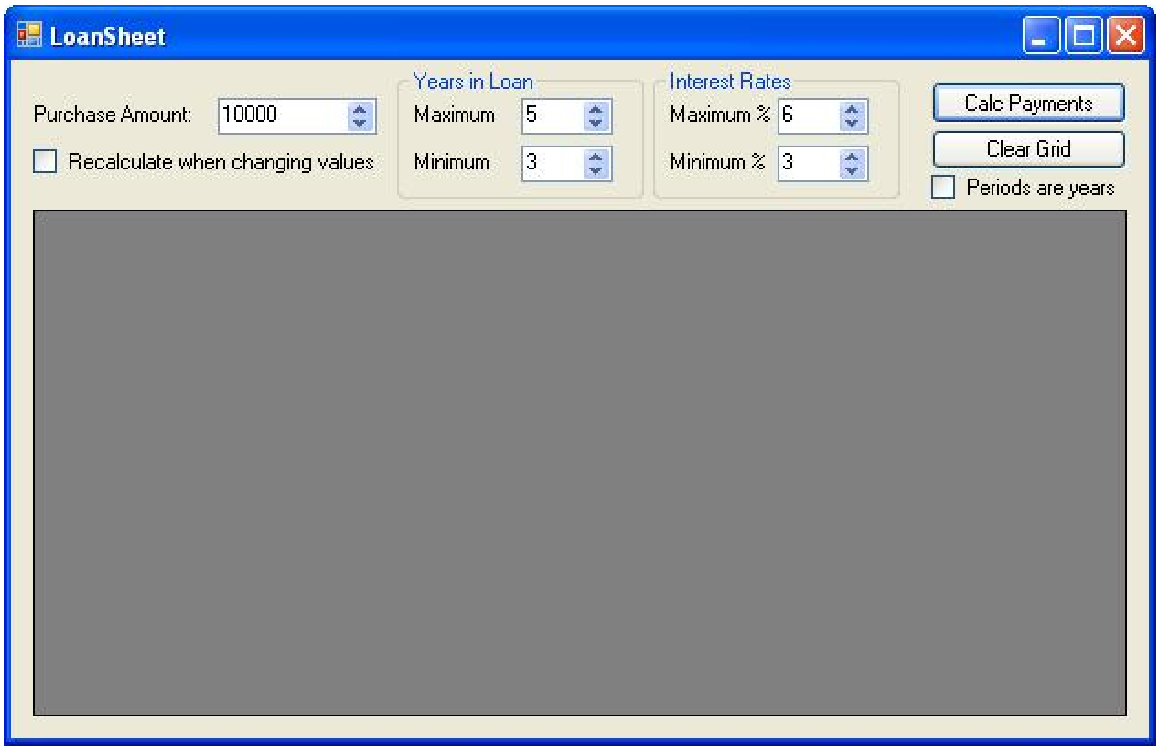
From: www.lescasse.com

Publisher: Unknown Publisher

While applications from the Internet can be useful, they can potentially harm your computer. If you do not trust the source, do not run this software. More Information...

Click the **Run** button.

After a short while the **LoanSheet** Dyalog Client-Server application should get displayed[3]:



LoanSheet

Purchase Amount: 10000	Years in Loan	Interest Rates	
	Maximum 5	Maximum % 6	Calc Payments
☐ Recalculate when changing values	Minimum 3	Minimum % 3	Clear Grid
			☐ Periods are years

As you can see, this is a real Windows application.

---

[3]If you are getting the following error when trying to load/Run the application:

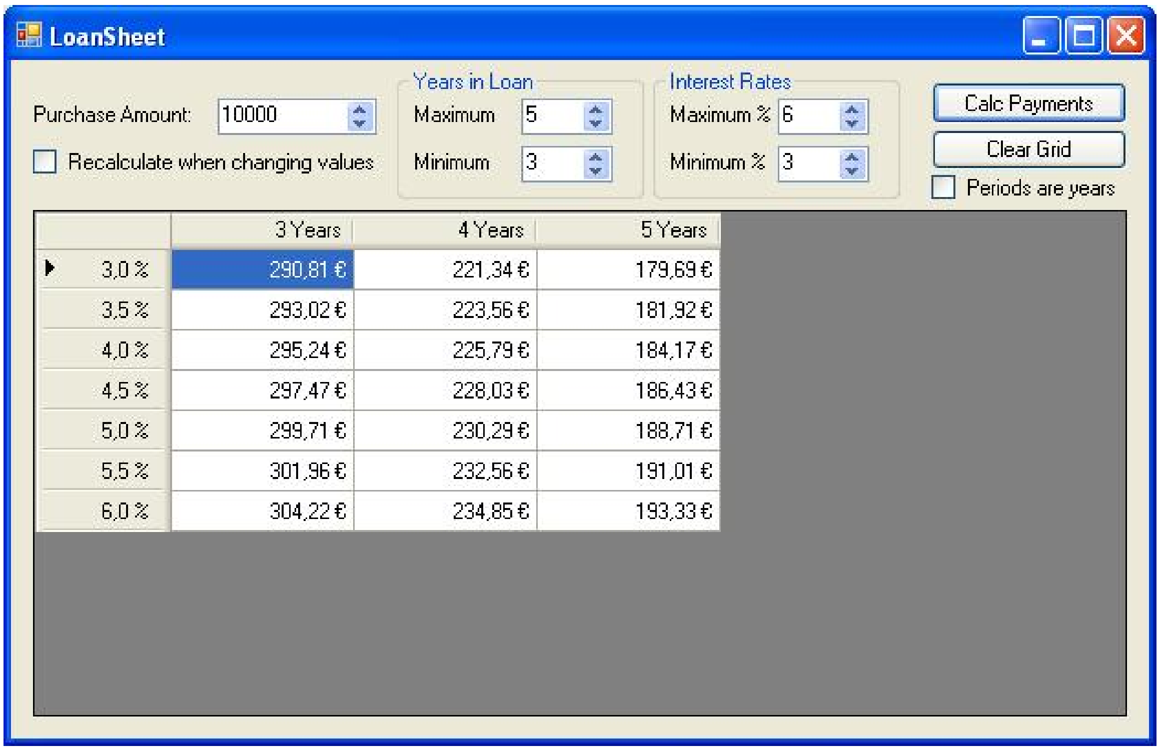
```
The remote server returned an error: (407) Proxy Authentication Required
```

this is due to a known Microsoft .Net Framework Bug for ClickOnce applications, for which a hotfix is available (look at: http://support.microsoft.com/kb/917952 to request and install this hotfix)

It has started on your computer without any files being installed on your computer.

You can now close Internet Explorer, if you wish.

Click the **Calc Payments** button: you should see the following values being displayed in the DataGridView object:



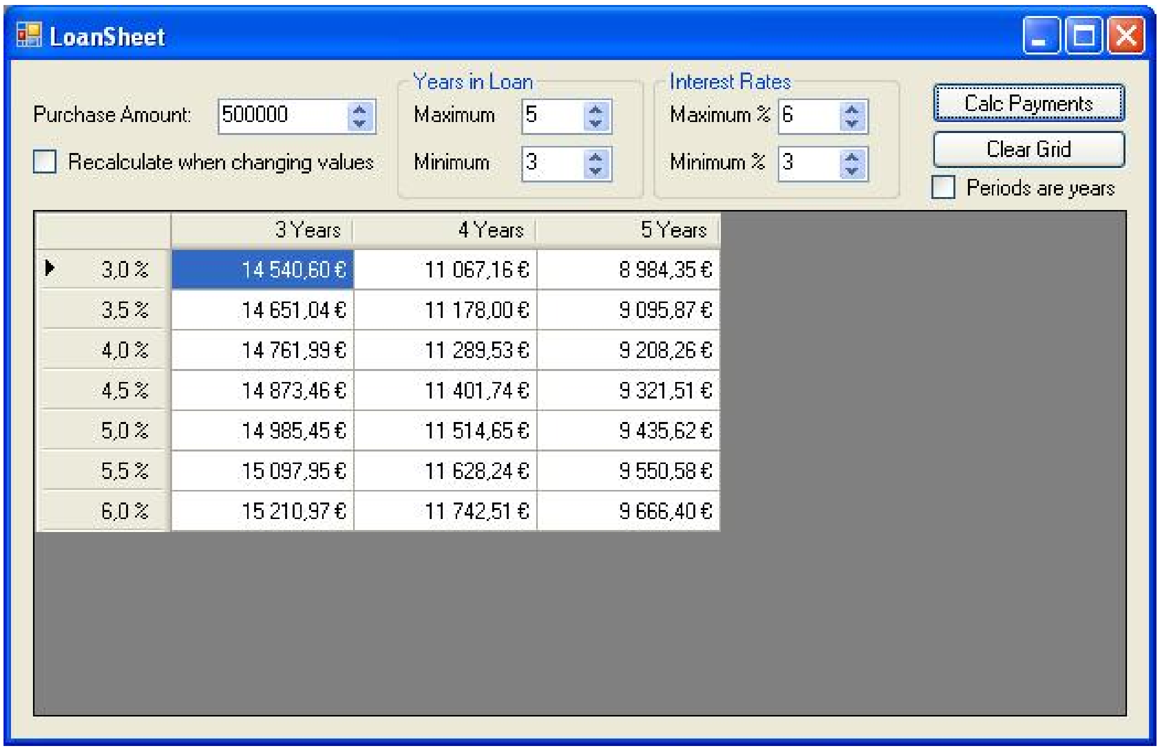
Purchase Amount: 10000

Years in Loan: Maximum 5, Minimum 3

Interest Rates: Maximum % 6, Minimum % 3

	3 Years	4 Years	5 Years
3,0 %	290,81 €	221,34 €	179,69 €
3,5 %	293,02 €	223,56 €	181,92 €
4,0 %	295,24 €	225,79 €	184,17 €
4,5 %	297,47 €	228,03 €	186,43 €
5,0 %	299,71 €	230,29 €	188,71 €
5,5 %	301,96 €	232,56 €	191,01 €
6,0 %	304,22 €	234,85 €	193,33 €

Change the **Purchase Amount** to be 500000 and click the **Calc Payments** button again.



Purchase Amount: 500000

☐ Recalculate when changing values

Years in Loan — Maximum: 5, Minimum: 3

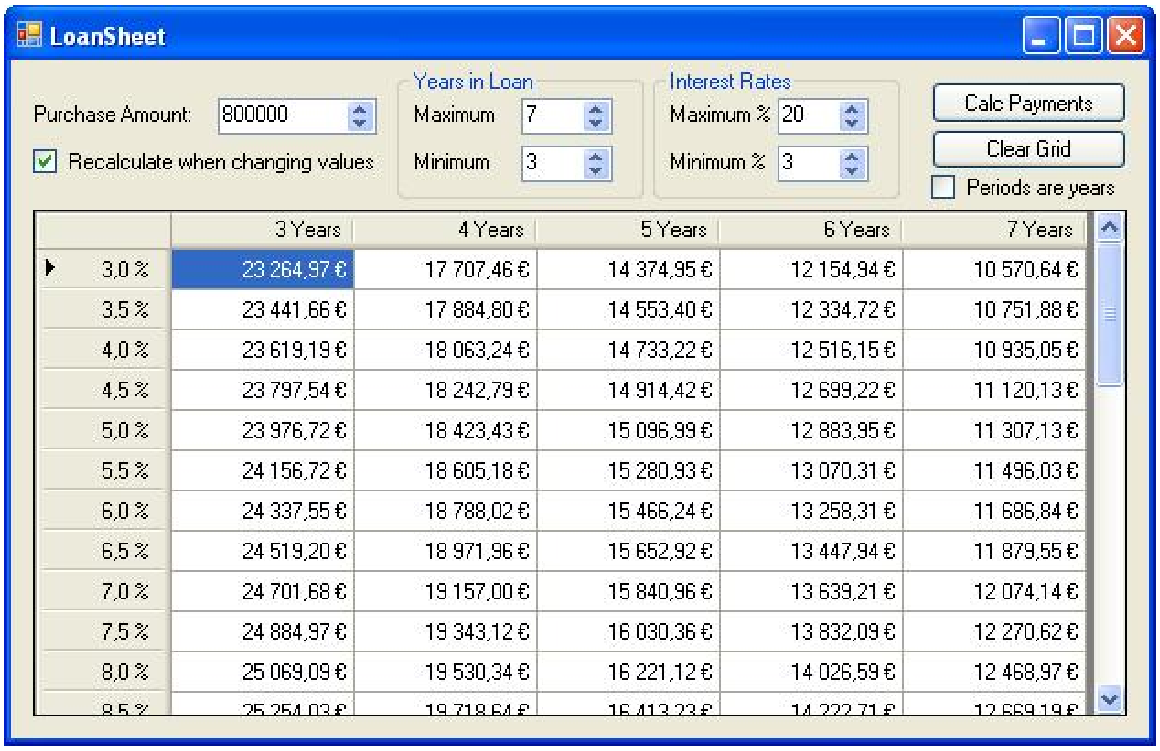
Interest Rates — Maximum %: 6, Minimum %: 3

☐ Periods are years

	3 Years	4 Years	5 Years
3,0 %	14 540,60 €	11 067,16 €	8 984,35 €
3,5 %	14 651,04 €	11 178,00 €	9 095,87 €
4,0 %	14 761,99 €	11 289,53 €	9 208,26 €
4,5 %	14 873,46 €	11 401,74 €	9 321,51 €
5,0 %	14 985,45 €	11 514,65 €	9 435,62 €
5,5 %	15 097,95 €	11 628,24 €	9 550,58 €
6,0 %	15 210,97 €	11 742,51 €	9 666,40 €

You may even check the **Recalculate when changing values** check box and then use the little buttons in the NumericUpDown edit boxes to change the **Purchase Amount**, **Years in Loan** or **Interest Rates**.

As you change any of these values the grid is recalculated and displayed.



Purchase Amount: 800000

☑ Recalculate when changing values

Years in Loan — Maximum: 7, Minimum: 3

Interest Rates — Maximum %: 20, Minimum %: 3

☐ Periods are years

	3 Years	4 Years	5 Years	6 Years	7 Years
3,0 %	23 264,97 €	17 707,46 €	14 374,95 €	12 154,94 €	10 570,64 €
3,5 %	23 441,66 €	17 884,80 €	14 553,40 €	12 334,72 €	10 751,88 €
4,0 %	23 619,19 €	18 063,24 €	14 733,22 €	12 516,15 €	10 935,05 €
4,5 %	23 797,54 €	18 242,79 €	14 914,42 €	12 699,22 €	11 120,13 €
5,0 %	23 976,72 €	18 423,43 €	15 096,99 €	12 883,95 €	11 307,13 €
5,5 %	24 156,72 €	18 605,18 €	15 280,93 €	13 070,31 €	11 496,03 €
6,0 %	24 337,55 €	18 788,02 €	15 466,24 €	13 258,31 €	11 686,84 €
6,5 %	24 519,20 €	18 971,96 €	15 652,92 €	13 447,94 €	11 879,55 €
7,0 %	24 701,68 €	19 157,00 €	15 840,96 €	13 639,21 €	12 074,14 €
7,5 %	24 884,97 €	19 343,12 €	16 030,36 €	13 832,09 €	12 270,62 €
8,0 %	25 069,09 €	19 530,34 €	16 221,12 €	14 026,59 €	12 468,97 €
8,5 %	25 254,03 €	19 718,64 €	16 413,23 €	14 222,71 €	12 669,19 €

As you can see, the application reacts pretty quickly to recalculate the loan and to display the results whenever you click a button.

However, here is what happens each time you click a button in this application:

- The various values displayed in the **LoanSheet** controls (except the ones displayed in the grid) are read from the user interface and sent to my Internet Server which is located in Paris, France, as a nested vector

- A **CalcPayments** Dyalog method is called in a Dyalog .Net object installed on my Server

- A floating point matrix is returned from the Dyalog method call to the Client application and displayed in the grid.

If this raised your interest, the next sections gives a few hints on what is required to create such a Client-Server .Net Dyalog application.

Creating a Client-Server .Net Dyalog application

Step 1: Creating the Dyalog .Net DLL

First, to do so, you need Dyalog 11 since you will need to create Dyalog objects.

Let's assume you have written a Dyalog application and let's assume this application is the LOAN.DWS workspace as distributed with the latest versions of Dyalog APL.

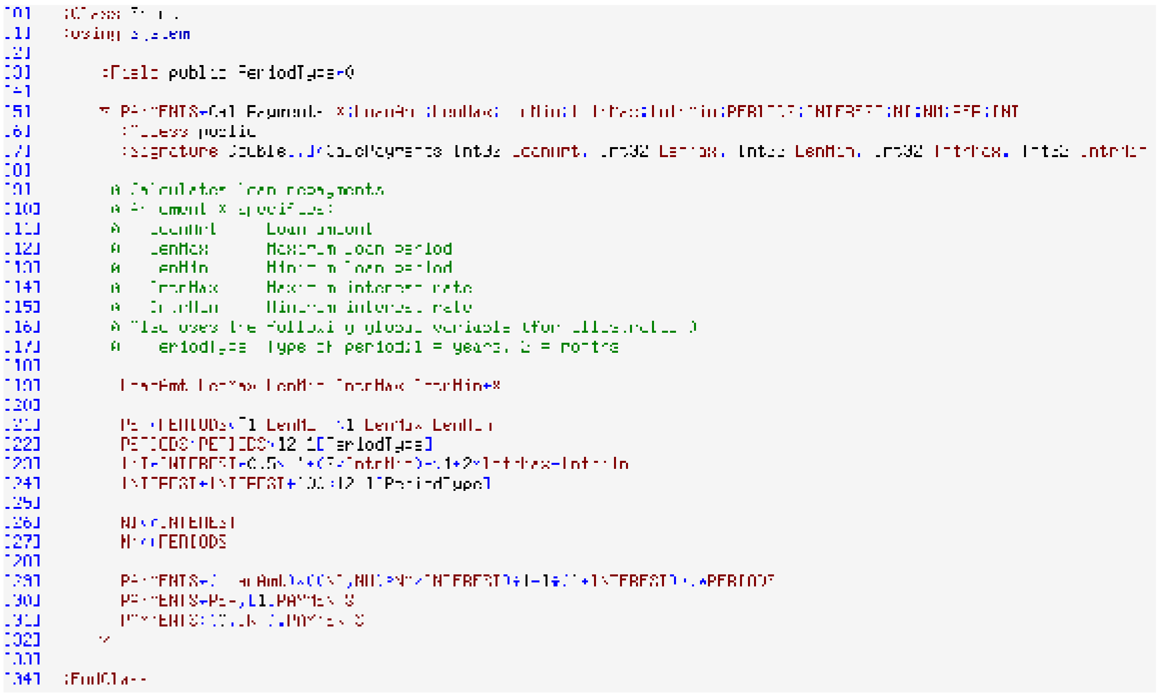
The first step is to create a Dyalog .Net DLL out of your application.

You can do that by loading the **LOAN.DWS** workspace, going to the **Loan** namespace and encapsulating the functions and variables you want to publish in your .Net DLL into a Dyalog 11.0 class.

```
      )load loan
C:\Program Files\Dyalog\Dyalog APL 11.0\samples\ole\loan
saved Fri Mar 03 11:21:36 2006

      )cs Loan
#.Loan
```

Then create a class, for example called **Sheet**, with one field called **PeriodType** and one method called **CalcPayments**, as follows:



```
:Class Sheet
:using System

    :Field public PeriodType←0

    ∇ PAYMENTS←CalcPayments X;LoanAmt;LenMax;LenMin;IntrMax;IntrMin;PERIODS;INTEREST;NI;NM;PER;INT
      :Access public
      :Signature Double[,]←CalcPayments Int32 LoanAmt, Int32 LenMax, Int32 LenMin, Int32 IntrMax, Int32 IntrMin

     ⍝ Calculates loan repayments
     ⍝ Argument X specifies:
     ⍝   LoanAmt      Loan amount
     ⍝   LenMax       Maximum loan period
     ⍝   LenMin       Minimum loan period
     ⍝   IntrMax      Maximum interest rate
     ⍝   IntrMin      Minimum interest rate
     ⍝ Also uses the following global variable (for illustration)
     ⍝   PeriodType   Type of period:1 = years, 2 = months

      LoanAmt LenMax LenMin IntrMax IntrMin←X

      PERIODS←¯1+LenMin+⍳1+LenMax-LenMin
      PERIODS←PERIODS×12 1[PeriodType]
      INTEREST←0.5×¯1+(2×IntrMin)+⍳1+2×IntrMax-IntrMin
      INTEREST←INTEREST÷100×12 1[PeriodType]

      NI←⍴INTEREST
      NM←⍴PERIODS

      PAYMENTS←LoanAmt×((NI,NM)⍴NM/INTEREST)÷1-1÷(1+INTEREST)∘.*PERIODS
      PAYMENTS←PERIODS,[1]PAYMENTS
      PAYMENTS←(0,INTEREST),PAYMENTS
    ∇

:EndClass
```

The important points here are:

- Make a public field out of any global variable you need to support as a property in your Dyalog .Net DLL

- Create a method with the same name as any function in your application that you want to support as a method in your Dyalog APL.Net DLL

- Be sure to declare the methods **public** with the `:Access public` declaration statement

- Be sure to provide the appropriate signature for every method you include in your class. Here we used:

  ```
  :Signature Double[,]←CalcPayments Int32 LoanAmt,
  Int32 LenMax, Int32 LenMin, Int32 IntrMax, Int32
  IntrMin
  ```

  which means that the **CalcPayments** method is returning a matrix of doubles (=floating point numbers in APL terms) and uses 5 arguments, all integers

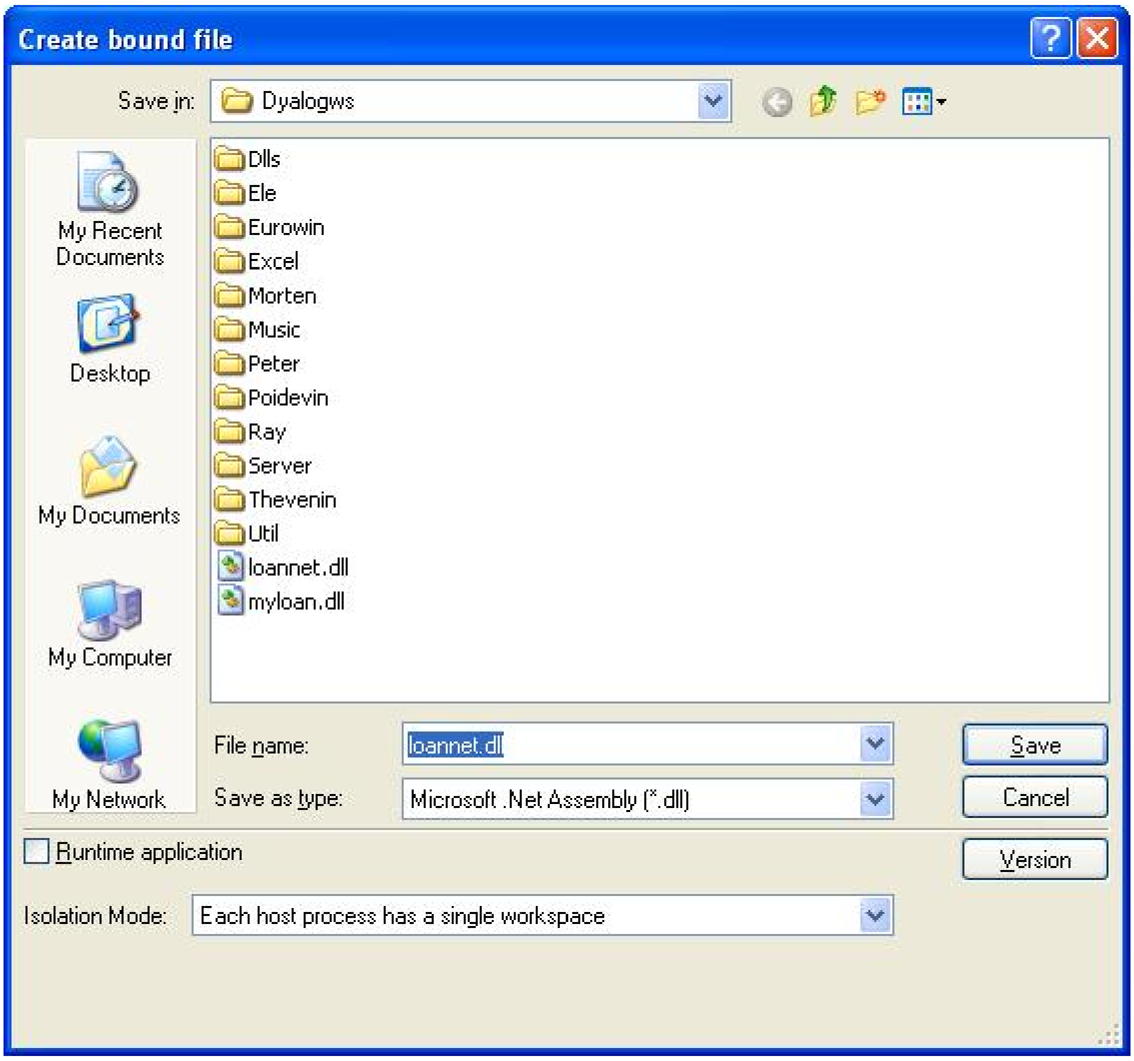
Once you have created your **Sheet** class, you need to register **Loan** as an OLE Server, following the steps described in the **Dyalog Version 11.0 Interface Guide pp. 251-253**

Then you may save the workspace under a new name to avoid altering the **LOAN.DWS** workspace:

```
      )wsid c:\dyalogws\loannet
was C:\Program Files\Dyalog\Dyalog APL
11.0\samples\ole\loan

      )save
. . .
```

Finally, to create the Dyalog .Net **LOANNET.DLL** DLL, select **File / Export** and fill the **Create Bound File** dialog as follows:



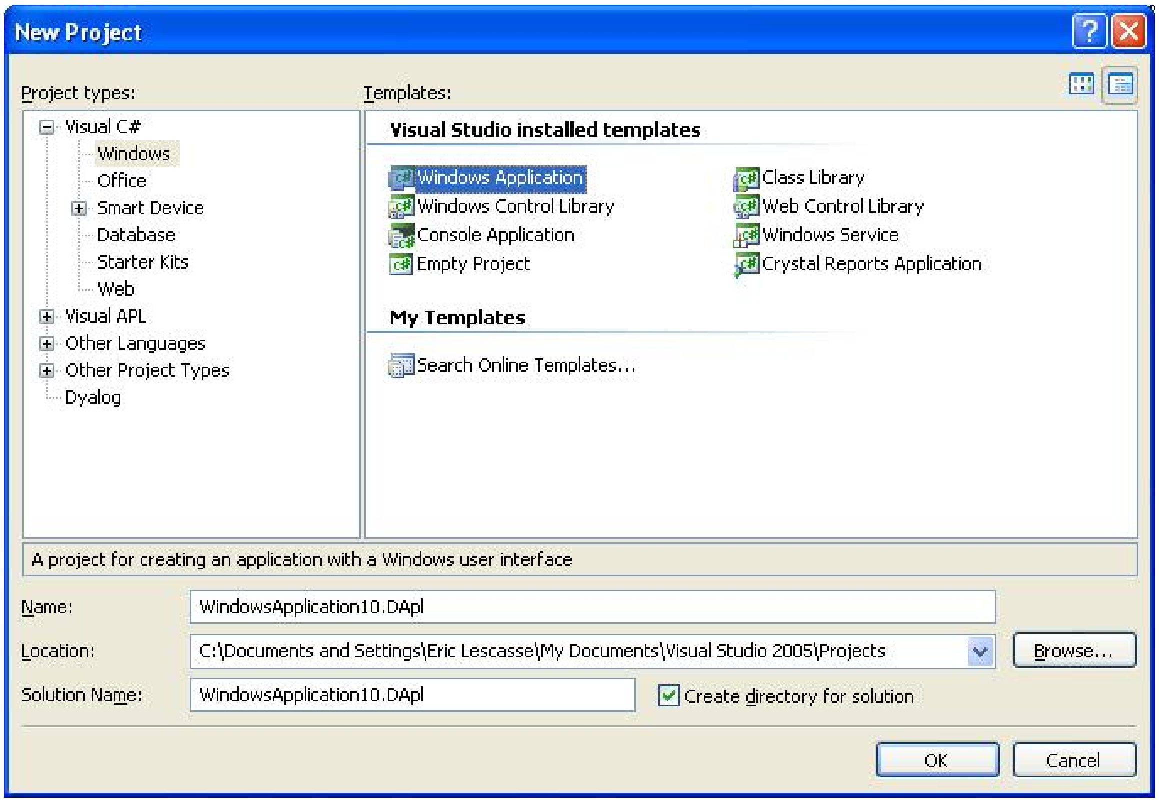
File name: loannet.dll
Save as type: Microsoft .Net Assembly (*.dll)
☐ Runtime application
Isolation Mode: Each host process has a single workspace

Step 2: Creating the Application User Interface

You need Visual Studio 2005 and C# to create the User Interface of your application.

Do not be afraid however since this is really easy, more than you may think of, even if you do not know C#.

Start Visual Studio 2005, select **File / New Project** and fill the dialog as follows:



A project for creating an application with a Windows user interface

Name: WindowsApplication10.DApl

Location: C:\Documents and Settings\Eric Lescasse\My Documents\Visual Studio 2005\Projects

Solution Name: WindowsApplication10.DApl

[✓] Create directory for solution

You can select any name you want for the project and for the Solution. I usually use the same name for both. You can include dots in the names as in the above example.

Once the project is created, you'll see the empty application form being displayed in Visual Studio.

It is out of the scope of this article to describe in detail how to build the .Net User Interface of our Loan application[4], but basically, you drag objects from the Toolbox displayed in the left hand side of Visual Studio 2005 and drop them on the form.

You can then move these objects, resize them, align them and change some of their properties at design time by using the **Properties** pane at the bottom right of Visual Studio 2005.

To create an event handler in C#, you most often simply double click on the object for which you want to handle an event. For example if you double click on the **Calc Payments** button, this will open the Form1.cs source code for your form and will automatically create the Click event handler for the **Calc Payments** button.

[4] **If you are interested to learn how to build .Net C# User Interfaces, you may want to look at the following on line Training:**

http://www.lescasse.com/CSharpTraining.asp

The first Screencast is available for free to every one and may give you enough information to get started. If you want to go further and access the other on line C# Training Sessions, please contact me at eric@lescasse.com

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

You can then add code between the curly brace characters.

Step 3: Create a C# wrapper class for the Dyalog Loannet.DLL

You should then add a class to your C# project, which will serve as a simple wrapper class around the Dyalog **LOANNET.DLL**.

Here is the code of this class:

```
    public class RemoteApl : MarshalByRefObject
    {
        private Sheet loan = new Sheet();

        public int PeriodType
        {
            get { return (int)loan.PeriodType; }
            set { loan.PeriodType = (int)value; }
        }

        public double[,] CalcPayments(int i1, int i2, int i3, int i4, int i5)
        {
            return (double[,])loan.CalcPayments(i1, i2, i3, i4, i5);
        }
    }
```

For this class to work you need to Add a Reference in the C# project to the **LOANNET.DLL** Dyalog .Net DLL and to add the following using statement at the top of the project:

```
using Loan;
```

The above class is very simple. It starts by creating an instance of the **Sheet** APL class, which is called **loan**.

Then a C# property called **PeriodType** is defined. It simply calls the Dyalog **loan.PeriodType** field in the background.

Then a C# method called **CalcPayments** is defined: it simply calls the Dyalog **loan.CalcPayments** method in the background.

Step 4: Calling the CalcPayments Dyalog method from the C# client application

Calling the Dyalog method from the C# Client application is pretty simple.

Just add the following code to the **button1_Click** event handler:

```
private void button1_Click(object sender, EventArgs e)
{
    RemoteApl apl = new RemoteApl();
    apl.PeriodType = 1+(checkBox1.Checked ? 1 : 0);
    double[,] res = apl.CalcPayments(
        (int)nudLoanAmt.Value,
        (int)nudLenMax.Value,
        (int)nudLenMin.Value,
        (int)nudIntrMax.Value,
        (int)nudIntrMin.Value
        );
}
```

This code is almost self explanatory.

It starts by creating a new instance called **apl** of the **RemoteApl** class that we defined in Step 3.

Then it sets the **PeriodType** property of this **apl** instance to **2** or **1** depending if the **Period are years** check box is checked (look at the screenshots at the beginning of this paper).

When the following instruction is executed:

```
apl.PeriodType = 1+(checkBox1.Checked ? 1 : 0);
```

the **PeriodType** property of the **RemoteApl** class is set. Setting this property executes the following code in the **RemoteApl** class:

```
set { loan.PeriodType = (int)value; }
```

which in turn changes the **PeriodType** field in the **loan** Dyalog class.

Finally, the **CalcPayments** method of the **RemoteApl** class is called, passing to it 5 arguments which are the values read from the various NumericUpDown objects displayed in our User Interface. Note that in C# an instruction can span over several lines: it is the final semi-colon which marks the end of the instruction. This is particularly handy to make programs readable, all the more than you may even add comments to each part of the instruction.

Example:

```
            double[,] res = apl.CalcPayments(
                (int)nudLoanAmt.Value,        // loan amount
                (int)nudLenMax.Value,         // maximum loan period
                (int)nudLenMin.Value,         // minimum loan period
                (int)nudIntrMax.Value,        // maximum interest rate
                (int)nudIntrMin.Value         // minimum interest rate
                );
```

Note that the **Value** property of a NumericUpDown object is a **decimal** in C# and needs to be cast to an integer, i.e. the **(int)** prefix to the various NumericUpDown objects.

It is essential that the parameters passed to the **CalcPayments** match both the declaration of the **RemoteApl CalcPayments** method and the data types defined in the Dyalog APL **loan.CalcPayments** method properties.

Then the **button1_Click** event handler captures the result returned from the **CalcPayments** Dyalog function in a 2-dimensional array of doubles called **res**.

From then on, you can use the **res** variable in C# to display the results in a DataGridView object.

I will not comment this additional code, but here is how you would do that, provided you would have added a DataGridView object called dataGridView1 to your form:

```
dataGridView1.SuspendLayout();
dataGridView1.Columns.Clear();
dataGridView1.DefaultCellStyle.Alignment =
            DataGridViewContentAlignment.MiddleRight;
DataGridViewCellStyle style = new DataGridViewCellStyle();
for (int i = 1; i < res.GetLength(1); i++)
{
      dataGridView1.Columns.Add("Col" + i.ToString(),
                  res[0,i].ToString() + " Years");
      style.Alignment = DataGridViewContentAlignment.MiddleRight;
      dataGridView1.Columns[i - 1].SortMode =
                  DataGridViewColumnSortMode.NotSortable;
      dataGridView1.Columns[i - 1].HeaderCell.Style = style;
}
object[] cells = new object[res.GetLength(1)-1];

for (int i = 1; i < res.GetLength(0); i++)
{
      for (int j = 1; j < res.GetLength(1); j++)
            cells[j-1] = string.Format("{0:C}", res[i, j]);
      dataGridView1.Rows.Add(cells);
}
dataGridView1.ResumeLayout();
```

This includes:

- dynamically creating the DataGridView columns since their number depends on values set in the interface,
- setting cell styles so that numeric values are right aligned in the cells and in the column header cells
- formatting data in all the cells
- populating the grid with the results sent by Dyalog APL

With just a little habit this is pretty simple code to write. The **SuspendLayout** and **ResumeLayout** methods are useful for avoiding displaying any change in the interface until the grid is completely populated and formatted.

Step 5: Adding the C# Remoting layers

At this stage, you might wonder how on earth it is possible for C# to call Dyalog objects properties and methods, after all.

Well, this is made possible by using .Net Framework Remoting.

It is out of the scope of this article to describe how this is done, but this means adding one more C# project which uses Remoting, installing the compiled version of this application on the Server as well as adding a few lines of code in the **Form1.cs** class to authorize communication with the C# Remoting layer.

The Server might be your local machine (represented by http://localhost) or a remote Server (represented by its IP address: http://xxx.xxx.xxx.xxx)

I should mention that such an application needs to use a port above port 1000.

For example the sample Client-Server Dyalog application which you can run from:

http://www.lescasse.com/dapl/publish.htm

uses port **10100** on my Server.

Step 6: Testing the application from Visual Studio

Once everything has been written, debugged and set up correctly, you can start using your Client-Server application.

To do so, click the Visual Studio **Start Debugging** button or press **F5**.

The application user interface should get displayed.

When you click on the **Calc Payments** button you should see the Dyalog **CalcPayments** method result be displayed in the interface.

If you have checked the **Recalculate when changing values** check box and then change some values in the NumericUpDown objects and click **Calc Payments** button again, you'll see a new grid of results being displayed, all calculated in the background by Dyalog APL.

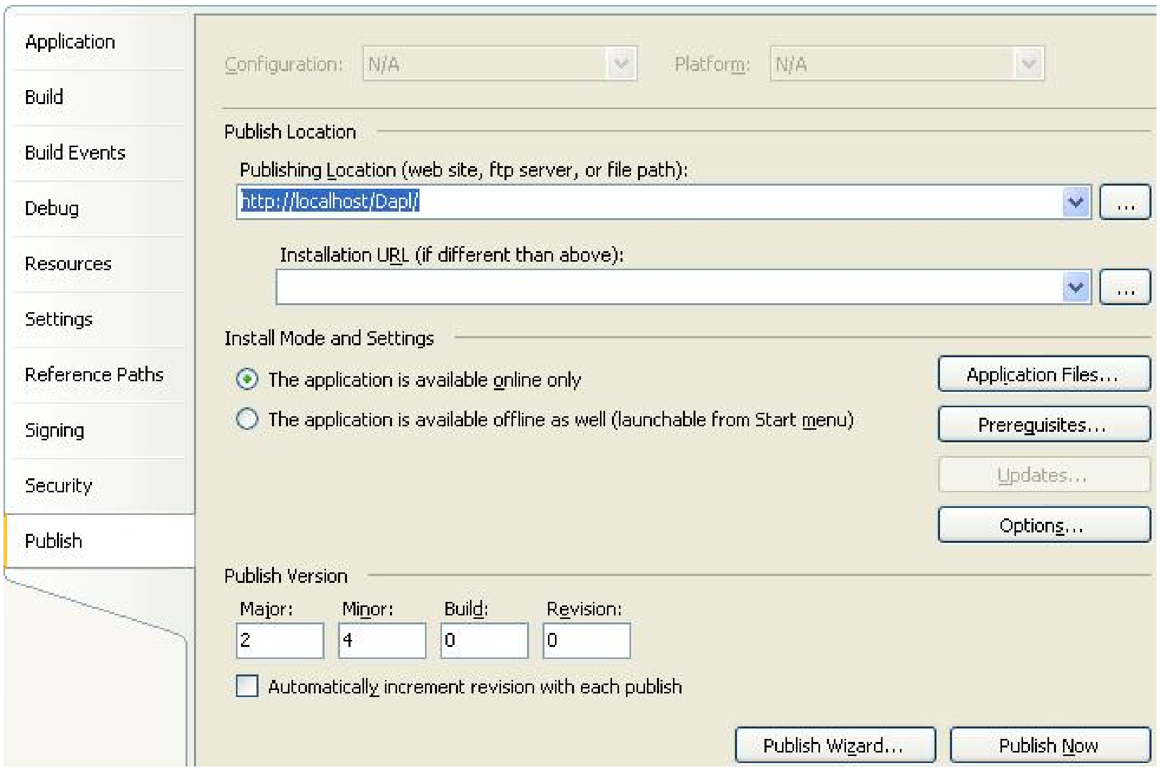
Step 7: Deploying the application locally

We are now reaching maybe the most important (and easy) step of the whole process: deploying the application to the Server as a **ClickOnce** application.

This is simply done by double clicking **Properties** under the project name in **Solution Explorer** in Visual Studio 2005.

The application **Properties** get displayed in Visual Studio

Click on the **Publish** Tab and fill it as follows:



Configuration: N/A Platform: N/A

Publish Location

Publishing Location (web site, ftp server, or file path): http://localhost/Dapl/

Installation URL (if different than above):

Install Mode and Settings

(●) The application is available online only

( ) The application is available offline as well (launchable from Start menu)

Publish Version

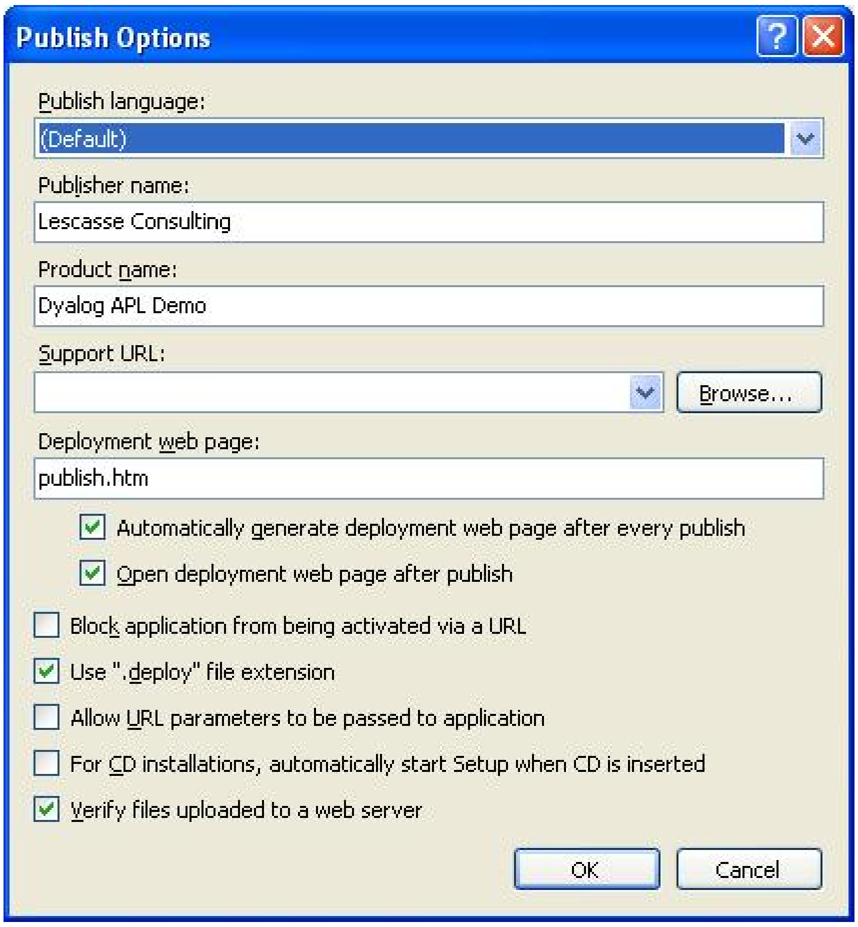
Major: 2 Minor: 4 Build: 0 Revision: 0

[ ] Automatically increment revision with each publish

The important points here are:

- The publishing location: enter: http://localhost/Dapl for example
- Check the radio button called **The application is available online only**

You may also enter a **Publish Version** number and may want to click on **Options** to set a few additional options, for example:



Publish language: (Default)

Publisher name: Lescasse Consulting

Product name: Dyalog APL Demo

Support URL:

Deployment web page: publish.htm

[✓] Automatically generate deployment web page after every publish

[✓] Open deployment web page after publish

[ ] Block application from being activated via a URL

[✓] Use ".deploy" file extension

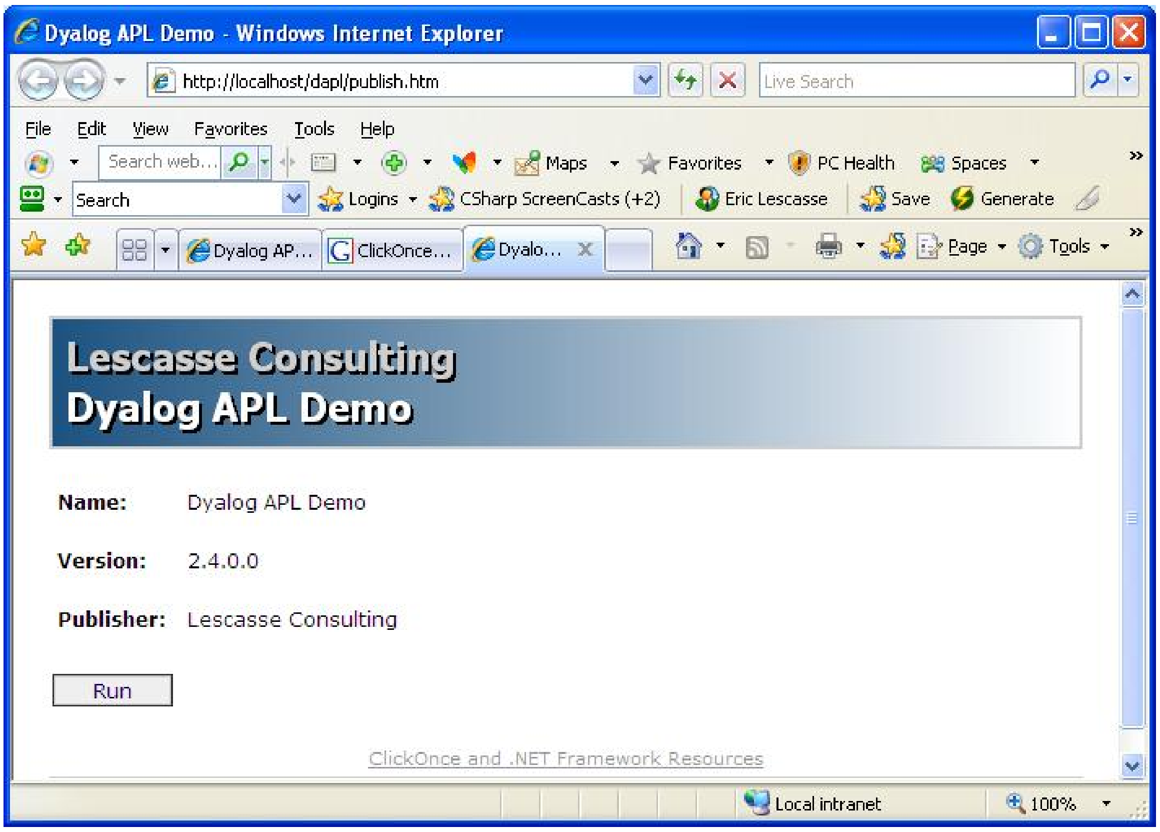
[ ] Allow URL parameters to be passed to application

[ ] For CD installations, automatically start Setup when CD is inserted

[✓] Verify files uploaded to a web server

Then simply click the **Publish Now** button.

After a short while, if everything compiles ok, Visual Studio will start Internet Explorer and display the published page allowing to launch the application:


Dyalog APL Demo - Windows Internet Explorer
http://localhost/dapl/publish.htm
File Edit View Favorites Tools Help
Lescasse Consulting
Dyalog APL Demo
Run
ClickOnce and .NET Framework Resources
Local intranet
100%

Name: Dyalog APL Demo

Version: 2.4.0.0

Publisher: Lescasse Consulting

From there, you may click the **Run** button to start the application from Internet Explorer and click **Run** again in the next dialog to confirm.

Step 8: Deploying the application on a Remote Server

Now that you can run the Client-Server Dyalog application locally (i.e. your development computer serves as both the Client computer and as the Server), you may want to install the application on a real Remote Internet Server.

This means:

1. Copying the remoting layer that you compiled on your local machine to the Server
2. Properly registering this remoting layer
3. Changing http://localhost to http://xxx.xxx.xxx.xxx in your **Form1.cs** application (where xxx.xxx.xxx.xxx is the IP Address of the Server)
4. Recompile the application under Visual Studio 2005
5. Republish the application (as shown in Step 7)
6. Copying the Published version of the application to the Server

Now, if all these steps have been done correctly, you should be able to call the application from any computer, anywhere, provided that you have an Internet access.

Just start Internet Explorer and enter the following URL:

http://xxx.xxx.xxx.xxx/dapl/publish.htm

where xxx.xxx.xxx.xxx is the IP address of the Server or its name.

For example, you may try my sample Client-Server Dyalog demo from:

http://www.lescasse.com/dapl/publish.htm

Pre-Requisites, Security & Performance Issues

First, clients should have a recent enough computer for such an application to be able to run. That means it is recommended they run Windows XP SP2, or Windows 2003 or Windows Vista.

It is also recommended they run Internet Explorer 6 or 7.

Such a Client Server application will not run directly under FireFox, though you can launch it from FireFox and FireFox will automatically start Internet Explorer for you and then run the application.

The client must have the **Microsoft .Net Framework 2.0** installed on his computer: however the ClickOnce application automatically detects the absence of the .Net Framework 2.0 and automatically installs it if it is not there. A reboot might be needed before being able to successfully load the application in such a case.

Finally, it sometimes happen that the Client runs a strong Firewall and in that case it may be necessary for him to authorize the port used by the Client-Server application in this Firewall.

Among all the people I know who tried :

http://www.lescasse.com/dapl/publish.htm

I have only encountered a couple of clients unable to load and run this demo and their problems could be solved quickly.

When publishing such a Client Server Dyalog application, it is essential that you have totally secured the Dyalog code which runs on the Server and if possible used error handling to avoid having an error stopping the program in Dyalog APL.

This would be very bad, since the Client C# application would suddenly become totally unresponsive and would have to be killed.

Also, as far as resources are concerned on the Server, it seems that the .Net Dyalog DLL is shared by the various client users. This means that such a Dyalog Client-Server could be used by a large number of users without impairing the Server performance.

Other than that, anyone can verify by running the sample demo I am presenting here from my Web site that such a Client-Server Dyalog application is very efficient. You can see that by clicking the **Recalculate when changing values** check box and then repeatedly clicking on the **Purchase Amount** NumericUpDown buttons to increase or decrease the amount. Just remember that each time you click one of these buttons:

- All values are read from the interface
- They are sent to the **CalcPayments** Dyalog method running on my Server in Paris, France
- The Dyalog **CalcPayments** method is run on my Server
- The results are sent back to you (i.e. the Client C# part of the application)
- A DataGridView object is set up and populated with the results

And all this is almost instantaneous! However my Server is serving more than 10 Web Sites including a couple of large ones and currently runs about 10 such Client-Server APL applications, so without being too busy, it does have other tasks to deal with.

Conclusion

In this paper we have succinctly and not exhaustively explained how anyone can transform his Dyalog desktop application to become a Client-Server ClickOnce Internet application.

The advantages of doing so are numerous:

- The application becomes immediately usable by anyone, anywhere, provided the user has an Internet access and a browser

- The application can be loaded and started on the Client computer without any file installed on his computer (provided he fulfills the prerequisite of having the .Net Framework 2.0 available on his machine)

- If the .Net Framework 2.0 is not present on the Client computer, it is installed automatically (if user accepts it) when the application is first run

- The application is an Internet application

- However the application remains a Rich Client application (i.e. a pure Windows application)

- The application is a .Net application

- The application is a Client-Server application which means that some files maybe shared by ALL users (i.e. on the Server, including APL component files) and some other files maybe used on the local Client computer (including Dyalog component files created by the application)

- With such an application, there is no more need for any Installation Program, CD/DVD to send to customers, files to download and install by customers, etc.

- Delivering a new application update to all customers is simple and immediate: just upload the new set of application files to the Server and all of a sudden all clients use the new version

- It is of course possible to protect application usage in various ways, including Login/Password forms when application loads, etc.

- It is possible to create a pay by usage mechanism in such an application, etc.

Dyalog users who would like to port their Dyalog application to the Web using this technology may contact me.

It is no doubt that such powerful Client-Server applications will start to quickly emerge in the near future: the fact we can get advantage of the processing power of APL on the Server is certainly a plus for us.