WINDOW-BASED INTERFACES WITH WINLIKE AND ASP.NET

by Tobias Soppa

There are currently two established schools of thought on how to structure HTML interfaces: with *tables* or with *frames*. Now a further concept is added with *WinLIKE*, with which it is possible to program with windows in web browsers, just like under Windows. As the ServerControls and CodeBehind concepts of ASP.NET have changed the Web world somewhat, the purpose of this article is to show how ASP.NET and WinLIKE fit together and enable the creation of fast and flexible user surfaces without much effort.

Interface Concepts in the Web

Web pages today typically contain a large HTML page, which represents different areas with the help of a table: usually the header, the menu, two columns for the contents and the footer. Each action by the user usually leads to a complete reload of the page, even if the greater part of the page is not changed. Though this is annoying to the user and puts the server under stress, there is a plausible reason: Google & friends. Search engines are highly optimized regarding the administration of found content, but usually do not want to have anything to do with more complex interfaces with Frames or JavaScript. A simple link to an indexed file has to be able to reconstruct the entire Website structure again.

With enterprise applications, e.g. portals, ERP- or Workflow systems, front-end indexing search engines do not matter. For this reason the more flexible frames, which limit the resource usage, can be used without problems. However, this solution does not completely satisfy the beleaguered developer, because larger applications usually consist of numerous different framesets. After a user action, a frame must typically update one or more other frames, or even worse, sometimes has to create these in the first place, and that with consideration of the most diverse status information. Even a dedicated frame handler can help only a little bit and not offer the flexibility which one generally expects from windows.

Windows have the simple yet great advantage, that through their ability to overlap each other they remain independent in size and position. The limited space of a screen can therefore be used at any time to show the required content, independently of the content underneath it. This makes it easier not only for the developer, but also for the user, who can more easily get familiar with the surface and who can arrange and store the interface in the way that appears most meaningful to him. The personalization of an application is thereby guaranteed intrinsically as each user opens exactly the windows he or she needs automatically, without the developer ever having thought about it in advance.

How Does WinLIKE Work?

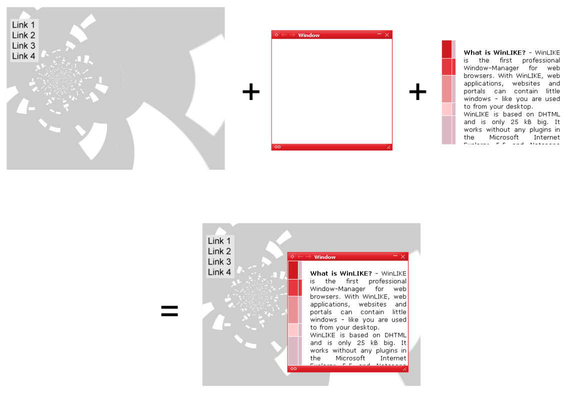
With WinLIKE, the familiar window world is now also available again for Web developers. WinLIKE is a Window Manager based exclusively on JavaScript and the Document Object Model (DOM) of IE and Mozilla. It is thus pure client technology, which does not need Plug-ins, ActiveX or JavaBeans. For this reason, WinLIKE can harmonize with any language on the server, for example ASP or ASP.NET. But let's take it one step at a time.

The first question that arises is what WinLIKE windows are in the first place. In principle, WinLIKE windows behave like Frames:
- Links with the target clause are loaded into the appropriate window
- Forms can send data via *get* or *post* to other windows
- Window content is completely encapsulated from other windows
- The window size can behave relatively or absolutely to the size of the browser
- Windows can be manipulated on the client with JavaScript
- Window definitions or actions can be generated on the server like frames

Windows, however, do not have the disadvantages of Frames:
- Window content is directly accessible over a Deeplink
- Windows are flexible and not as static in size and position as framesets
- Unlike framesets, windows can overlap themselves freely
- If necessary, windows can be created automatically

The similarity to frames is not coincidental, since WinLIKE windows are essentially composed of frames (in particular of *inline frames*), which were however refined with lots of JavaScript.

Generally speaking, a website or web application with windows thus consists of three components: 1. the root page, which for example contains the menu and the background, 2. several windows, which are defined in the root page and 3. the content, which is loaded into windows:



What is WinLIKE? - WinLIKE is the first professional Window-Manager for web browsers. With WinLIKE, web applications, websites and portals can contain little windows - like you are used to from your desktop.
WinLIKE is based on DHTML and is only 25 kB big. It works without any plugins in the Microsoft Internet

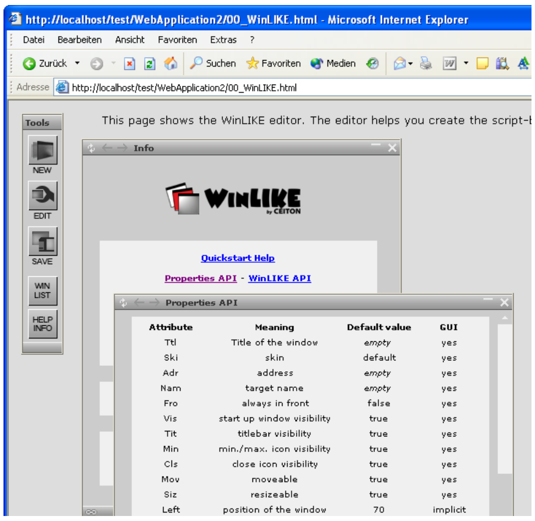
Fig. 1 - Structure of a WinLIKE page

The JavaScript source code of a root page (whether .html or .aspx file) looks as follows, if two windows are to be defined therein:

```
<SCRIPT SRC="winlike/winman/wininit.js"></SCRIPT>
<SCRIPT SRC="winlike/winman/winman.js"></SCRIPT>
<SCRIPT>
  WinLIKE.definewindows=mydefs;
  function mydefs()
  {
    var j=new WinLIKE.window('',130,45,460,270,15);
    j.HD=false;
    j.Adr='samples/asp.net/list.aspx';
    j.Nam='listwin';
    j.Ski='xp-png';
    WinLIKE.addwindow(j);

    var j=new WinLIKE.window('',230,187,430,280,19);
    j.HD=false;
    j.Vis=false;
    j.Nam='editwin';
    j.Ski='xp-png';
    WinLIKE.addwindow(j);
  }
</SCRIPT>
<BODY onLoad=WinLIKE.init()>
</BODY>
```

At the start the necessary WinLIKE resources are loaded and thereafter the two windows are defined. The first window, named listwin (.Nam='listwin'), is visible right after launching the page and loads a file (Adr='samples/asp.net/list.aspx'). The second window *editwin* is not visible (.Vis=false) and therefore does not need to load content yet (.Adr). By the way, the individual characteristics of a window are described in the *Properties API* of WinLIKE:


http://localhost/test/WebApplication2/00_WinLIKE.html - Microsoft Internet Explorer

This page shows the WinLIKE editor. The editor helps you create the script-

Quickstart Help

Properties API - WinLIKE API

Properties API

Attribute	Meaning	Default value	GUI
Ttl	Title of the window	*empty*	yes
Ski	skin	default	yes
Adr	address	*empty*	yes
Nam	target name	*empty*	yes
Fro	always in front	false	yes
Vis	start up window visibility	true	yes
Tit	titlebar visibility	true	yes
Min	min./max. icon visibility	true	yes
Cls	close icon visibility	true	yes
Mov	moveable	true	yes
Siz	resizeable	true	yes
Left	position of the window	70	implicit

Fig. 2 - Description of characteristics of a WinLIKE window

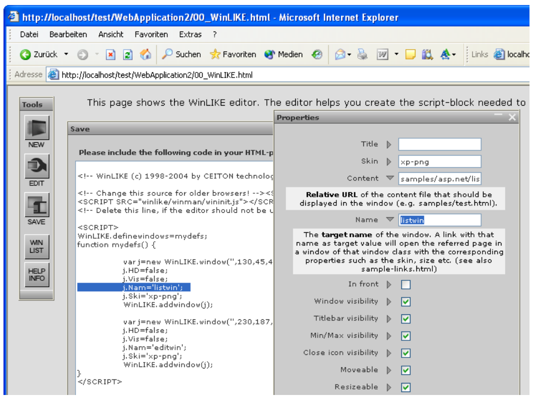
In the *onLoad* function of the body tag WinLIKE is then started (WinLIKE.init), so that the windows can be generated. That's it - the WinLIKE page is finished. The following link would then load the file *edit.aspx* into the window *editwin*:

```
<A HREF="samples/asp.net/edit.aspx?id=10263" TARGET="editwin">My
Link</A>
```

or with JavaScript:

```
<SCRIPT>WinLIKE.openaddress('samples/asp.net/edit.aspx',
'?id=10263', 'editwin');</SCRIPT>
```

One can generate windows easily more easily than by script with the WinLIKE editor (*00_WinLIKE.html* from the WinLIKE distribution), which is also explained in detail in the two *files01_tutorial_general.html* and *02_tutorial_content.html*. With the editor one generate windows visually by Drag-and-Drop and provides these with the necessary characteristics. A click on the *Save* button produces the required code for the window definitions, which can then simply be pasted into a root page:


http://localhost/test/WebApplication2/00_WinLIKE.html - Microsoft Internet Explorer

This page shows the WinLIKE editor. The editor helps you create the script-block needed to

Save

Please include the following code in your HTML-p

```
<!-- WinLIKE (c) 1998-2004 by CEITON technolog

<!-- Change this source for older browsers! --><S
<SCRIPT SRC="winlike/winman/wininit.js"></SCR
<!-- Delete this line, if the editor should not be u

<SCRIPT>
WinLIKE.definewindows=mydefs;
function mydefs() {

        var j=new WinLIKE.window('',130,45,4
        j.HD=false;
        j.Vis=false;
        j.Nam='listwin';
        j.Ski='xp-png';
        WinLIKE.addwindow(j);

        var j=new WinLIKE.window('',230,187,
        j.HD=false;
        j.Vis=false;
        j.Nam='editwin';
        j.Ski='xp-png';
        WinLIKE.addwindow(j);
}
</SCRIPT>
```

Properties

Title	
Skin	xp-png
Content	samples/asp.net/lis

Relative URL of the content file that should be displayed in the window (e.g. samples/test.html).

Name	listwin

The **target name** of the window. A link with that name as target value will open the referred page in a window of that window class with the corresponding properties such as the skin, size etc. (see also sample-links.html)

In front	☐
Window visibility	✓
Titlebar visibility	✓
Min/Max visibility	✓
Close icon visibility	✓
Moveable	✓
Resizeable	✓

Fig. 3 - WinLIKE editor with the generated window definitions

An ASP.NET example

In order to learn how WinLIKE cooperates with the ASP.NET ServerControls and the CodeBehind feature, we use an example to show how a Listview is loaded into a window and how the entries can be manipulated in a new window. To achieve this, the two tables *Orders* and *Customers* from the *Northwind* database are used, which for this example must run on an SQL server or an MSDE:

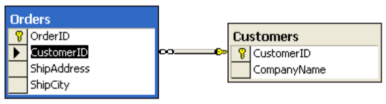
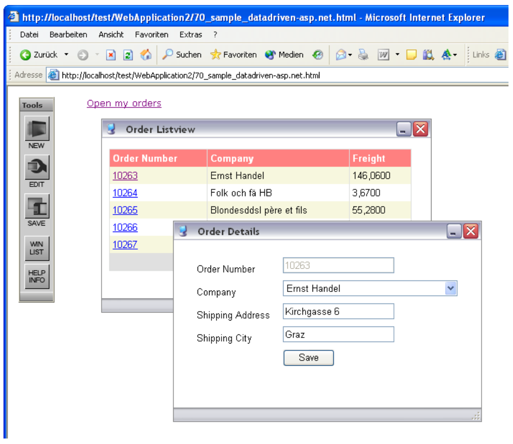


Fig. 4 - The database schema

On completion, the small example application should appear as follows:


http://localhost/test/WebApplication2/70_sample_datadriven-asp.net.html - Microsoft Internet Explorer

Open my orders

Order Listview

Order Number	Company	Freight
10263	Ernst Handel	146,0600
10264	Folk och fä HB	3,6700
10265	Blondesddsl père et fils	55,2800
10266		
10267		

Order Details

Order Number	10263
Company	Ernst Handel
Shipping Address	Kirchgasse 6
Shipping City	Graz

Save

Fig. 5 - The example application in a Web Browser

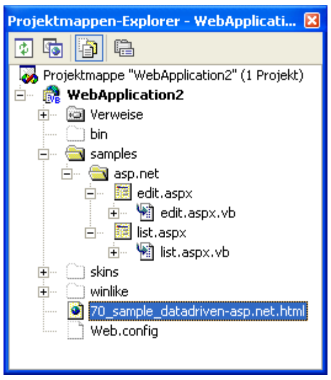
The example, which is a component of the WinLIKE distribution, does not need Visual Studio.NET, but a directory on a web server. If, however, the example is to be loaded into Visual Studio, the CodeBehind files must be called differently. The *src* attribute of

```
<%@ Page Language="vb" src="list.aspx.vb" Inherits=".list" %>
```

must be replaced by the CodeBehind attribute:

```
<%@ Page Language="vb" CodeBehind="list.aspx.vb" Inherits=".list"
%>
```

The project thus consists of the root page *70_sample_datadriven-asp.net.html* and the two window content files *list.aspx* and *edit.aspx* together with the appropriate CodeBehind files:



Projektmappen-Explorer - WebApplicati...

```
Projektmappe "WebApplication2" (1 Projekt)
WebApplication2
  Verweise
  bin
  samples
    asp.net
      edit.aspx
        edit.aspx.vb
      list.aspx
        list.aspx.vb
  skins
  winlike
  70_sample_datadriven-asp.net.html
  Web.config
```

Fig. 6 - Visual Studio Project Map

When working with Visual Studio.NET, a second point is to be considered. By default, the WinLIKE root page displays two pictures the size of the browser window when starting:

```
<BODY onLoad='WinLIKE.init()'...
<!-- Don't remove this line!--><IMG ID=...
<!-- You can change this loading picture! --><IMG ID=...
```

These serve informational purposes (load.gif) and for deactivating links (trans.gif) as long as WinLIKE is not completely loaded yet. Whilst working on the files in the visual editor, these pictures are clearly in the way because they are in the foreground and block visual editing. During development, the two pictures can be turned into comments temporarily.

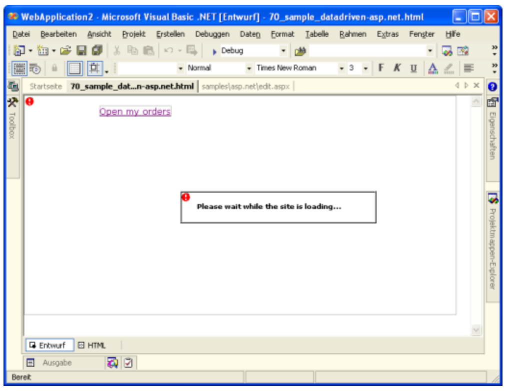
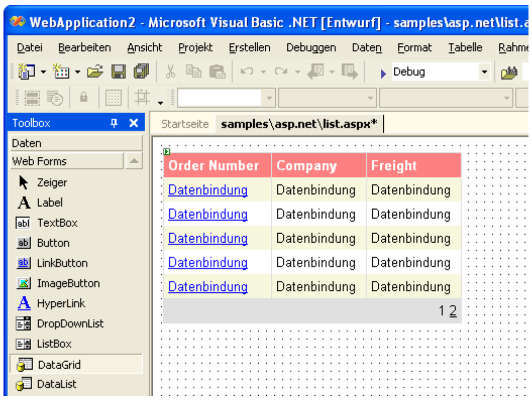

WebApplication2 - Microsoft Visual Basic .NET [Entwurf] - 70_sample_datadriven-asp.net.html
Open my orders
Please wait while the site is loading...
Entwurf
HTML
Bereit

Fig. 7 - WinLIKE root page in the draft mode of Visual Studio

As already discussed, the structure of the root page consists only of the two window definitions. Now the actual ASP.NET files must be designed. For the Listview the standard *DBGridControl* with *Paging* is used, for the time being without data connection. It can be pulled simply by Drag-and-Drop from the toolbox on the left side:


WebApplication2 - Microsoft Visual Basic .NET [Entwurf] - samples\asp.net\list.a
Datei Bearbeiten Ansicht Projekt Erstellen Debuggen Daten Format Tabelle Rahme
Debug
Toolbox
Daten
Web Forms
Zeiger
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
DataGrid
DataList
Startseite
samples\asp.net\list.aspx*

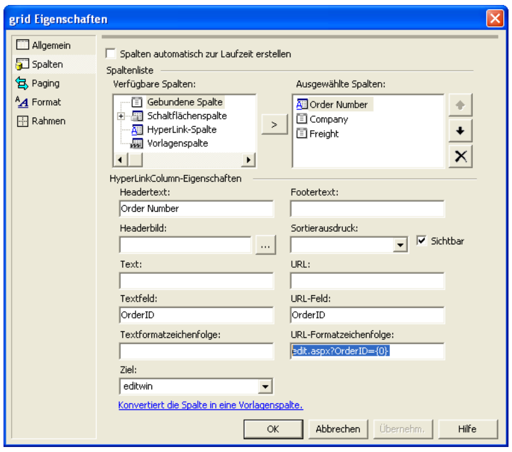
Order Number	Company	Freight
Datenbindung	Datenbindung	Datenbindung
Datenbindung	Datenbindung	Datenbindung
Datenbindung	Datenbindung	Datenbindung
Datenbindung	Datenbindung	Datenbindung
Datenbindung	Datenbindung	Datenbindung
		1 2

Fig. 8 - Draft view of the Listviews

Here the first column is important, as it contains the link through which the respective data record is then to be worked upon in a new window. In addition to the standard attributes such as HREF the link needs two pieces of information: 1. the target window and 2. the ID of the associated data record.
In the simplest case, the target window is defined simply by the target attribute of the link and the ID is passed on by *Querystring*. The link, which is to be generated by data record by the server control, should therefore look as follows:

```
<A HREF="edit.asp?OrderID=10263" TARGET="editwin">10263</A>
```

This is accomplished quite easily by adding a column of the type *hyperlink* in the control characteristics. To this column in the fields *URL formatting character sequence* and *goal* these two pieces of information can be transferred:



[ ] Spalten automatisch zur Laufzeit erstellen

Spaltenliste

Verfügbare Spalten: Gebundene Spalte, Schaltflächenspalte, HyperLink-Spalte, Vorlagenspalte

Ausgewählte Spalten: Order Number, Company, Freight

HyperLinkColumn-Eigenschaften

Headertext: Order Number	Footertext:
Headerbild:	Sortierausdruck: [✓] Sichtbar
Text:	URL:
Textfeld: OrderID	URL-Feld: OrderID
Textformatzeichenfolge:	URL-Formatzeichenfolge: edit.aspx?OrderID={0}
Ziel: editwin	

Konvertiert die Spalte in eine Vorlagenspalte.

Abb. 9 - Characteristics of the Listview in VS.NET

The source code thus generated looks like follows, where {0} is replaced at runtime by the contents of the 0th (respectively first) column of the dataset tied up later.

```
<asp:HyperLinkColumn
  DataNavigateUrlFormatString="edit.aspx?OrderID={0}"
  Target="editwin"
</asp:HyperLinkColumn>
```

The associated source code is likewise no more complicated. First the function *BindData()* calls the data up from the Northwind database and binds the data set produced to the *GridControl*. Through a *Join* in the Select statement, the ID references of the customer column are replaced by the correct names of the customer table:

```
Sub BindData()
  Dim sql As String = "SELECT o.OrderID ...
  Dim cn As New
  SqlConnection("server=localhost;database=northwind...
  Dim da As New SqlDataAdapter(sql, cn)
  Dim ds As New DataSet
  da.Fill(ds)
  grid.DataSource = ds
  grid.DataBind()
End Sub
```

Since Paging is activated in the Listview, the current Page pointer must be buffered when reloading the window, which for example must be done after the storage of a data record in the other window. In our example this is achieved via the session variable *Page*, which is read out during a Reload (thus no post back) and is assigned to the Listview again:

```
Private Sub Page_Load(ByVal...) Handles MyBase.Load
  If Not Page.IsPostBack Then
    If Session("Page") Then
      grid.CurrentPageIndex = Session("Page")
    End If
    BindData()
  End If
End Sub

Sub grid_Page(ByVal Sender...
  grid.CurrentPageIndex = E.NewPageIndex
  Session("Page") = E.NewPageIndex
  BindData()
End Sub
```

Thanks to ASP.NET, this small and simple piece of code suffices to fill the window with content and to generate the standard link to the second window.
Now this Edit window has nothing further to do than load and store the correct data record in order to subsequently update the Listview with the new values.
For the form, again the WebForm-Controls of the type System.Web.UI.WebControls are used, which can likewise be pulled into the page by Drag-and-Drop.

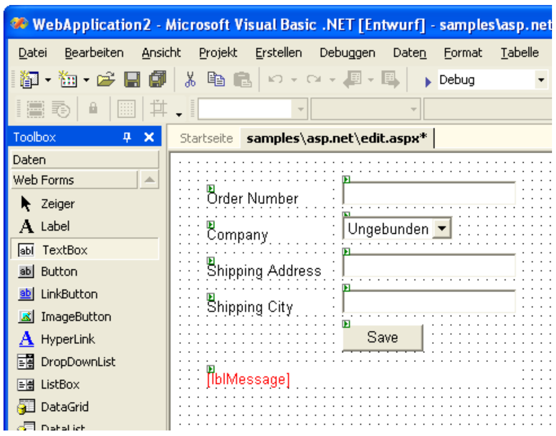


Abb. 10 - Draft view of the Edit form

The Combo box (DropDownList) is to indicate the customer names at runtime and uses the customer ID for clear identification (CustomerID):

```
<asp:textbox id="txtOrderID" runat="server"></asp:textbox>
<asp:dropdownlist id="dropCustomers" runat="server"
  DataValueField="CustomerID"
  DataTextField="CompanyName">
</asp:dropdownlist>
...
```

With normal JavaScript the focus is set on the combo box so that the form immediately gets the focus as soon as the window is loaded. The *Try-Catch* instruction is important to avoid an error in some browsers if the window is closed even before content has been loaded:

```
<SCRIPT>
  try
  {
    document.getElementById("dropCustomers").focus();
  }
  catch (everything) {}
</SCRIPT>
```

When calling the page now, first the form has to be filled with data. With the help of ID handed over by the Querystring the SQL statement is assembled. The inquiries here are realized not with a DataSet, but with a *Reader* object. In addition, the complete Customers Table is loaded in order to fill the Combo box. The pre-selected value of the box is then set naturally to that value, which corresponds to the data record from the first inquiry (read.Item(3)):

```
Private Sub Page_Load(ByVal...) Handles MyBase.Load
  cn = New   SqlConnection("server=localhost...
  If Not Page.IsPostBack Then
    LoadData()
  End If
End Sub

Sub LoadData()
  sql = "SELECT ... WHERE OrderID=" + _
                Request.QueryString("OrderID")
  cmd = New SqlCommand(sql, cn)
  cn.Open()
  read = cmd.ExecuteReader()
  read.Read()
  txtOrderID.Text = read.Item(0)
  txtShipAddress.Text = read.Item(1)
  txtShipCity.Text = read.Item(2)
  Dim remember = read.Item(3)
  read.close()

  sql = "SELECT CustomerID, CompanyName FROM Customers"
  cmd = New SqlCommand(sql, cn)
  dropCustomers.DataSource = cmd.ExecuteReader()
  dropCustomers.DataBind()
  cn.Close()
  dropCustomers.SelectedValue = remember
End Sub
```

With this, the form is correctly loaded and the only thing that remains is saving. As usual an *update* instruction is generated from the values returned automatically by PostBack an instruction for update provided and set off on the database server. The only thing to consider here is the injection of two WinLIKE API calls by *Response.Write*, with which in case of success the Listview is to be loaded freshly and the window closed again in the super ordinate browser DOM element (parent of the window = root page). In the event of an error only an error message is returned instead:

```
Private Sub btnUpdate_Click(ByVal...
  sql = "UPDATE ... SET CustomerID='" + _
          dropCustomers.SelectedValue + "'...
  cmd = New SqlCommand(sql, cn)
  Try
    cn.Open()
    cmd.ExecuteNonQuery()
    Response.Write("<SCRIPT>" + _
    "parent.WinLIKE.openaddress(...);" + _
    "parent.WinLIKE.windows[...].close();" + _
    "</SCRIPT>")
  Catch er As Exception
    lblMessage.Text = er.Message
  Finally
    cn.Close()
  End Try
End Sub
```

All in all the required effort is negligible, and the whole affair heavily reminiscent of Visual Basic under Windows. And that is what really counts. The developer should be able to concentrate on the concrete conversion of the typically complex business processes and functions and should be relieved from routine jobs and details with the help of prefabricated components.

Summary

It remains to be noted that it is quite easy to produce WinLIKE compatible code with the server-based controls of ASP.NET. Despite the considerable standardization it is possible to easily design window-based user interfaces because of the flexibility of the .NET Controls and the restriction of WinLIKE on HTML and JavaScript. The limitations, which have hampered Web developers up to the present, are now finally being removed to a large extent.

As a final hint, please look at the example *43_develop_submit_other.html* of the WinLIKE distribution. There it is shown, how not only individual data can be handed over to other windows with Querystrings, but also how served bases forms can post complete data records to other windows. On this note: End of the Posting and have fun Windowing with .NET!

Links:
WinLIKE: http://www.winlike.de
The here discussed sample is a part of the WinLIKE distribution.

Prices:
WinLIKE is free for non-commercial usages. A Single Server Licence costs 80 US$ and the Volume Server Licence between 75 and 40 US$ per installation.

Tobias Soppa is Managing Director of CEITON technologies GmbH and is occupied with Internet technologies, development systems and enterprise processes. He occasionally writes about those topics in the technical press and holds seminars at trade fairs and exhibitions.