Author: Chris Cohen
chris.cohen@vizola.com

Building a simple application using BCA

Project: Cash-flow Estimator

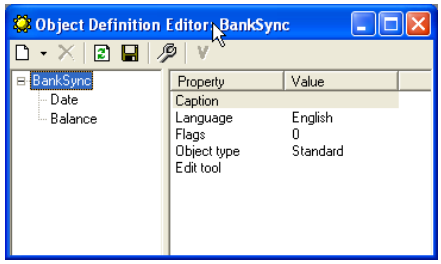
Table-of-Contents

Introduction

This application note takes you through the steps of creating a simple cash-flow estimator which generates a month-to-view report showing flows in and out of a bank account. For this project we want to be able to select the month of interest up to 6 months ahead and report on expected cash flows during the month.

Note: This is a simplified actual example from an ERP implementation, which leaves out expected cash flows from invoicing and purchasing sources.

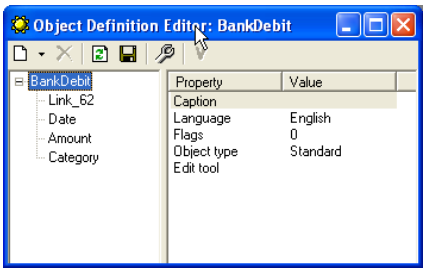
Synchronising the estimator



Every so often we want to be able to synchronise our model with an actual value, which takes account of any discrepancies between the simple model and reality. Our estimator will use only the most recent value as a starting point to project forward.

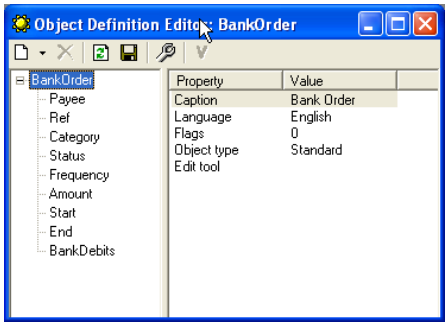
Using the BCA Object Definition Editor (ODE), we simply create a BankSync object which simply has two properties (Date, Balance).

Specifying regular credits/debits



Most bank accounts have fixed sums of money going in and out of the account on a regular basis which might be weekly, fortnightly, monthly or quarterly. These transactions could be broadly categorised into Wages/Salary (INCOMINGS) and overheads such as mortgage/rent/utilities/savings etc. as standing orders/direct debits (OUTGOINGS) which have a start- and end-date, frequency and amount. We want to be able to easily activate/deactivate each order so we'll have a status property. You can see that each bank order may represent a large number of individual transactions.

Whilst it would be possible to determine which are the relevant ones in the month of interest by parsing through at report generation time, this is not the best solution. It will be far faster to create individual transaction objects when we decide to activate the order, then simply query for ones in the month of interest when generating the report.



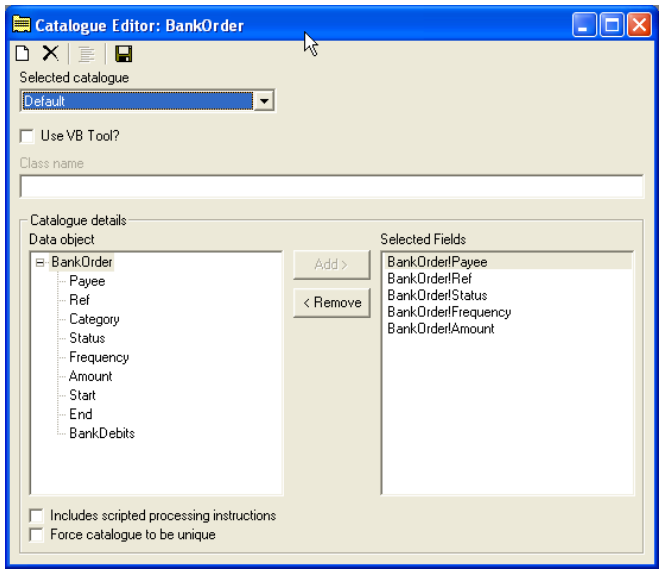
Using the ODE we therefore create this very simple BankDebit object with just three properties (Date, Amount, Category). Note: The ODE will automatically add another property which represents the MANY end of a ONE-MANY link.

We also create a BankOrder object which has the properties we've discussed plus a couple of other informational items and a special property (BankDebits) which we setup as a data object link to BankDebit as ONE-MANY.

Setting up the object properties for web page display

We have now created the three object definitions we need to implement the cash-flow estimator. In order to ensure that BCA knows how to display each of the object's properties we need to simply click on each one in turn and ensure that the appropriate data type is selected. For example, selecting CURRENCY right aligns the data defaulted to 0.00 and selecting DATE displays a field with popup calendar.

Setting up a Default catalogue for listing objects

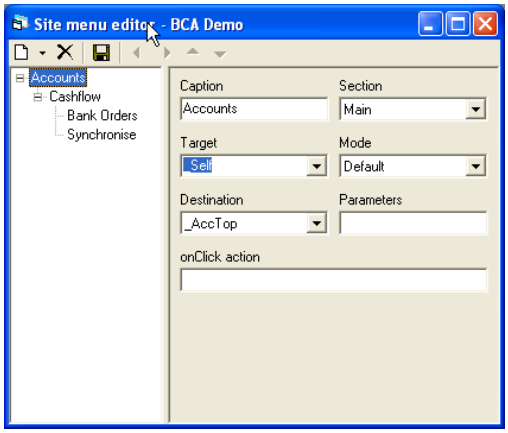


We also need to specify which properties to show in a listing row, which is done by creating a 'Default' catalogue for the object. The ODE catalogue editor allows us to simply create this and select the fields of interest.

Note: A BCA catalogue is analogous to an object-oriented index, allowing objects to be easily filtered and listed directly from the database without actually having to instantiate them.

For objects with specific requirements and complex interactions we can create different catalogues (scripted or compiled) and even force uniqueness.

Setting up a menu to access these objects

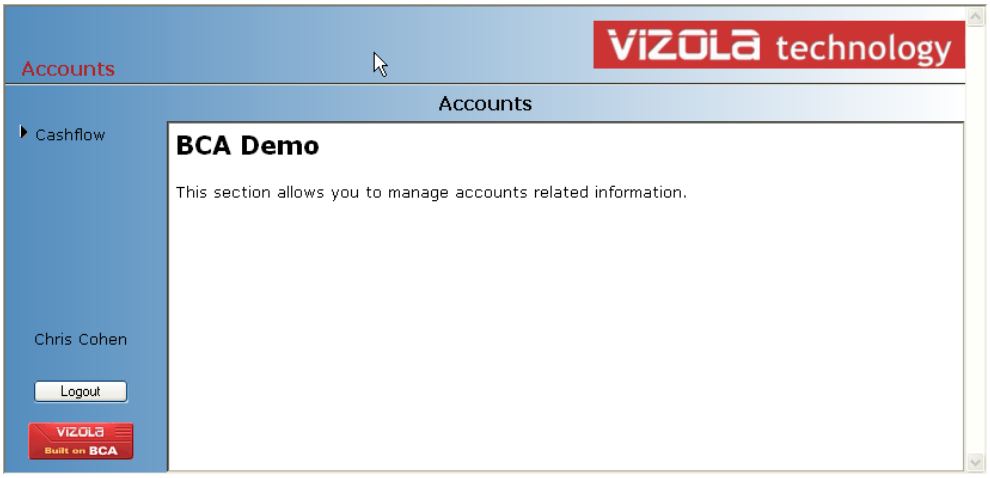


Once we have set up a default catalogue, we can start LISTING, ADDING and EDITING instances of these objects. But to access them we must first set up a few menu entry points with the aid of the ODE Site Menu Editor.

This tool allows you to construct a menu system for accessing aspects of your application. Note: In fact with BCA you can easily setup as many different sites and menus (including security sections) as you wish, with all pointed at the same database, which allows for creation of customer extranets etc.

For the sake of this project we can neatly access all aspects of the cash-flow estimator through three entry points:

1. The CASHFLOW menu item provides direct access to the report.
2. The BANK ORDERS item provides a listing of all these objects currently configured.
3. The SYNCHRONISE item lists bank synchronisation dates and amounts.



Note that the BCA convention is for root menu items to appear on the MainNav top stripe, and their child items appear in a LocalNav side pane. The BCA system only displays lower levels of menu hierarchy as you drill down into them.

For the purposes of demonstration we have here very quickly and easily added a new page object called _AccTop which merely displays the text you see here in the white 'working area'.

Note: The working area is bordered by the customisable BCA navigation and system areas defined in the site's MainPage.htm template and associated images and stylesheets.

Clicking on the CASHFOW menu item will drill down into active BCA pages, discussed below.

Returning to the site menu editor - fields in the right pane allow you to set properties for each item:

1. CAPTION is self-evident.
2. SECTION controls to which security section this item belongs.
   'Main' is generally setup for access by the group EVERYONE.
3. TARGET and MODE fields control whether the item appears in the standard window or a popup.
4. DESTINATION and PARAMETERS fields control which PAGE and what parameters should be called.

For this project, we will leave SECTION, TARGET and MODE at their default values, shown here, so the fields we do need to complete are DESTINATION and PARAMETERS.

We have already added a very simple plain HTML page as an introduction to the cash-flow estimator. For the other menu items we will use two very common *general purpose pages*, and an appropriate string of parameters.

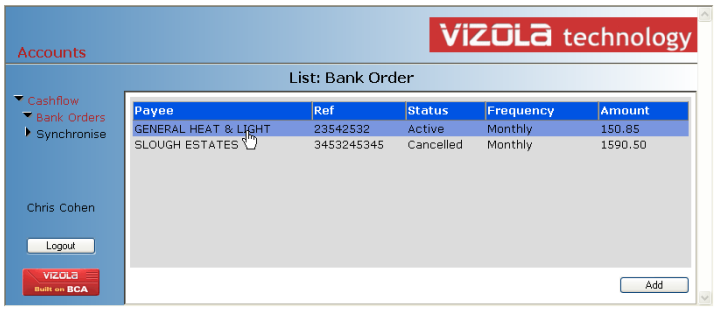
MENU ITEM	DESTINATION	PARAMETERS
Cashflow	Report	1,,,CashFlow,,Cash Flow Forecast
Bank Orders	CreateDO	0,BankOrder
Synchronise	CreateDO	0,BankSync

Note: We could set up a BCA system to use different names for these *general purpose pages* but the important thing is that they cause the system to instantiate classes which implement iclsHTMLTool and pass a set of (mostly optional) input parameters. CreateDO calls the default page handler class (bcaDOTools.HTMLForm) and Report calls the default report generator class (bcaReps.DoRep). By convention, the input parameters are in the same order as the default handler (Action, DefID, ObjID, ChildPath, ChildID, ParentNav, Flags), but this is only important when a custom handler does not implement all action types, whereupon program flow drops through to the default handler.

For the default page handler the first parameter is the ACTION and the second the DEFID.

ACTION = 0 means list object instances (catalogue list), and provides an entry point for manipulation of the objects:

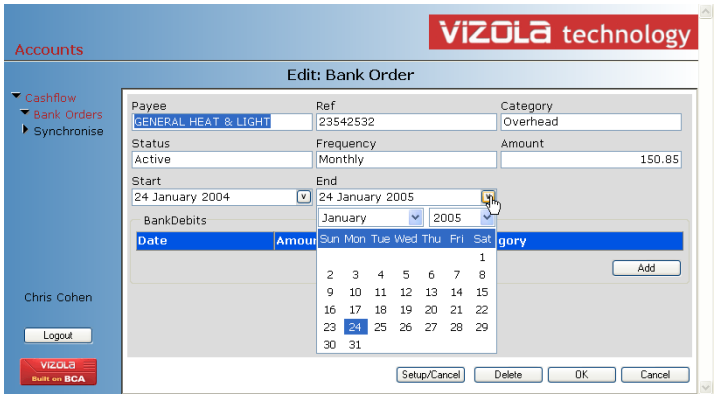
1. An ADD button is placed at the bottom of the page
2. Each row may be drilled down for editing the object



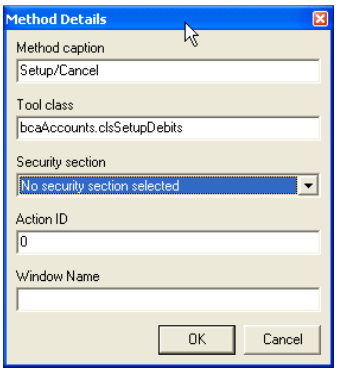
Upon drilling down we can see how BCA lays out the page fields for editing. Notice that AMOUNT has been defined as type CURRENCY so the field has been right aligned, whereas START and END are date type so they have been given a popup calendar.

Note also that since BankDebits is a one-many object link the system automatically generates a fieldset container for a catalogue of linked objects. In fact, since we don't really want to see any of this container we can set a HIDE=1 flag against BankDebits using the ODE.

You can also see the DELETE, OK and CANCEL buttons added by the editing page builder handler. You will notice also a fourth button SETUP/CANCEL which we have added as a custom 'method' in order to activate or de-activate bank orders by creating or deleting their linked BankDebits objects (see later section for details on how to do this).



Creating a custom SETUP/CANCEL method for BankDebits



We would like to do this so we can create and delete BankDebit objects simply by changing the status of its parent BankOrder.

Well, this time we need to write a bit of code – a simple COM object which implements an interface defined by iclsMethod in the BCA system. This interface module ensures the page handler inserts the method buttons when required (according to the optional configured parameters in the ODE) and provides access to the overall BCA DOManager object together with submitted NavParams.

By loading the BankOrder object of interest we can check its status and accordingly create child BankDebits if required, using the other property information. The code below illustrates use of some of the powerful (unrestricted) methods in the DOManager sub-objects.

Output for iclsMethod uses the ubiquitous HTMLReturn data type, using whose Page member we can ensure stateful onward navigation for the user, with optional error messages if required.

```
Option Explicit
'method button to setup/cancel bank debits according to bank order
Implements iclsMethod

Private Function iclsMethod_DoMethod(doMan As DOManager, Nav As Navigation, Params As ParamTool) As HTMLReturn
'use end points / frequency to create bank debits

Dim dtStart As Date, dtEnd As Date, dtDebit As Date, dtDebitActual As Date
Dim oBO As DataObj, oBD As DataObj
Dim lLink As Long, lParent As Long
Dim sErr As String
Dim str As New StrConcat
Dim aNavParams() As String

    aNavParams = Split(Nav.Params, ","): lParent = aNavParams(2) 'we know that ObjID is the third parameter

    'load the bankorder and find the children link number assigned by ODE
    Set oBO = doMan.DataObjectManager.Load(lParent): lLink = oBO.ItemInfo("BankDebits").LinkID

    doMan.DataObjectManager.Save DObject:=oBO, SubmitForm:=True 'first submit any changes to the object (from edit fields) and save it

    doMan.Hierarchies.RemoveChildren lLink, lParent 'delete any existing debits

    dtStart = oBO!Start: dtEnd = oBO!End: dtDebit = dtStart 'load up the bank order to get info

    Select Case oBO!Status
        Case "Active", "active"
            Do While dtDebit <= dtEnd 'this loop creates as many bankdebits as required
                dtDebitActual = missWeekend(dtDebit) 'make sure the actual debit date is a weekday
                Set oBD = doMan.DODefManager.CreateObj("BankDebit") 'create a BankDebit, copy the parent props and save
                With oBD: !Date = reverseDate(dtDebitActual): !Amount = oBO!Amount: !Category = oBO!Category: End With
                doMan.DataObjectManager.Save oBD
                doMan.Hierarchies.AddHierItem lLink, lParent, oBD.ObjectID 'object links are separated for rapid retrieval by query
                dtDebit = intervalAddByName(dtDebit, oBO!Frequency, dtEnd, sErr) 'add on the frequency
             Loop

        Case "Cancelled", "cancelled" 'we've already removed the previous children so do nothing

        Case Else: sErr = "Invalid status: Use Active or Cancelled." 'setup an error message if the user types an unknown status

    End Select

    'if err then first alert the error then set edit action again, else return to list
    If sErr <> "" Then str.Add "<script>alert('" & sErr & "');</Script>": aNavParams(0) = 1 Else aNavParams(0) = 0
    str.Add "<script>navigate('" & Nav.Target & "[" & Join(aNavParams, ",") & "]');</Script>"
    iclsMethod_DoMethod.Page = str.Text

End Function

Function reverseDate(dt As Date) As String
    reverseDate = Year(dt) & "/" & Right("00" & Month(dt), 2) & "/" & Right("00" & Day(dt), 2)
End Function

Function missWeekend(ByVal dt As Date) As Date
    Do While Weekday(dt) = 1 Or Weekday(dt) = 7: dt = dt + 1: Loop
    missWeekend = dt
End Function

Function intervalAddByName(dt As Date, interval As String, dEnd As Date, sError As String) As Date
Dim cnt As Integer
    cnt = 1
    Select Case interval
        Case "Monthly", "monthly", "m":         interval = "m"
        Case "Weekly", "weekly", "w":           interval = "ww"
        Case "Fortnightly", "fortnightly":      interval = "ww": cnt = 2
        Case "Quarterly", "quarterly", "q":     interval = "q"
        Case Else:                              interval = "d": dt = dEnd
                                                sError = "Invalid frequency: Use Weekly, Fortnightly, Monthly or Quarterly."
    End Select
    intervalAddByName = DateAdd(interval, cnt, dt)
End Function
```

Creating simple reports in the ODE

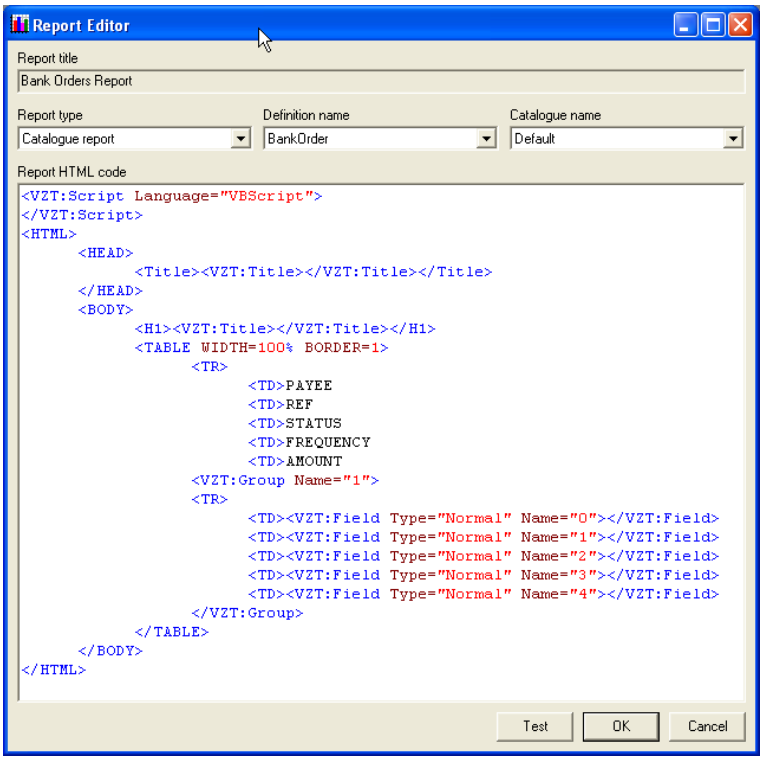
The ODE has an in-built HTML Report Editor which allows you to insert special `<VZT:Field>` tags into an HTML template of your design to populate it with relevant information.

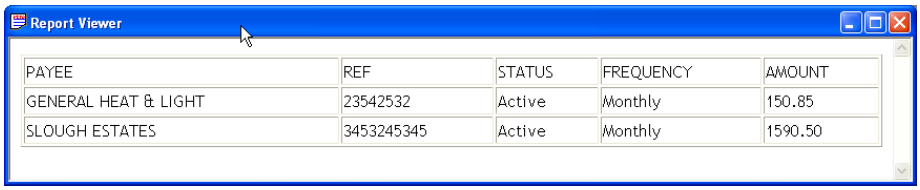
You simply choose the object of interest and then specify whether you want a summary (catalogue) report or a detail report on an individual object.

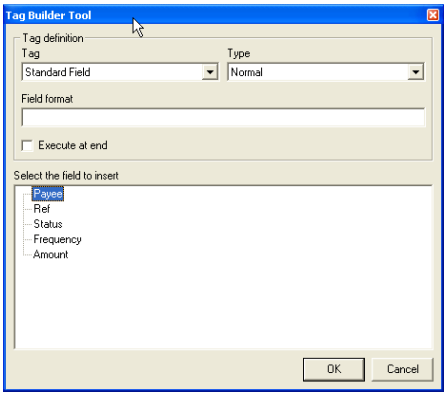
The report editor supports a `<VZT:Group>` tag which is used to iterate through either catalogue entries (as in the Bank Orders Report here), or sub-object collections (like purchase order lines) if you are reporting on a single object.

An important option on the context menu is the Tag Builder Tool which constructs the correct tag syntax for you when you simply select your tag type and field. Note that currently tags are named by ordinal position for catalogue reports.

A Test button allows you to instantly preview report output without setting up menu access within the application, which is the final step to prototyping our new report, as shown below.







Building flexible interactive reports

In addition to simple FIELD tags, the tag builder allows you to insert GROUP, TITLE, STORE and most importantly SCRIPT tags. Script tags allow full programmatic control of report building and are vitally important for constructing interactive reports, such as the month-to-view cash-flow report we are aiming towards.

You can think of the <VZT:Script> block as server-side script with access to local variables from the report context as well as some important system objects and classes, including:

- A parameters collection from the calling page or entity with important 'OpenArgs'
- An ADODB connection to the system database
- All ASP objects, including REQUEST object submitted fields
- BCA definition, object and catalog managers
- Various BCA utilities including an efficient string handler and listing tool

An HTML template for the Month-to-View Cash-flow Estimator

The layout for the template is really quite simple.

We need a select element populated with six months starting at the current month.

Then we need a table with a row for each day of the month, and appropriate column headers and footers.

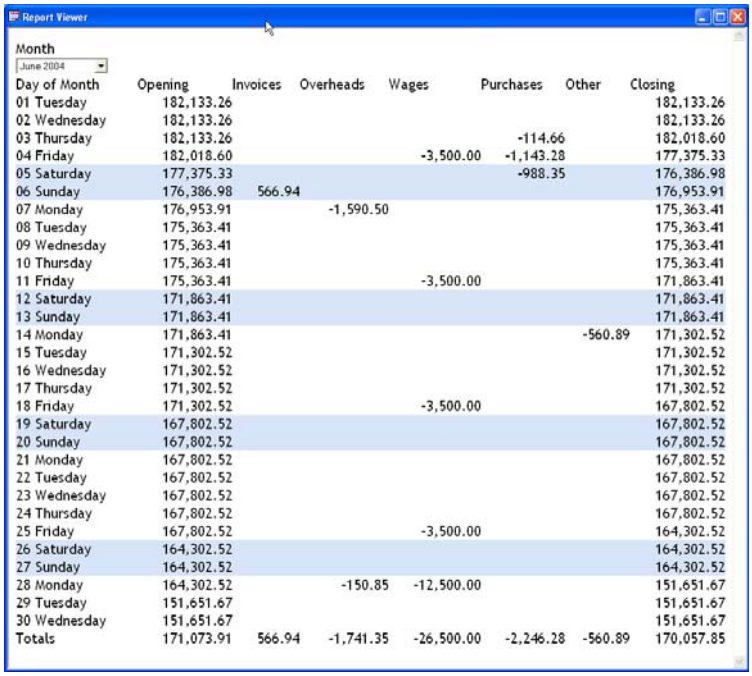
We will decide on the specific categories we wish to display in advance, and any other category transactions will be put within the OTHER column.

Building both the select element options and the table body will necessarily be done in script as there are several variables which govern the number of rows and information displayed.

Later, we will run through exactly what is involved in coding these scripts.

```
<HTML>
	<HEAD>
		<Title><VZT:Title>Cash Flow Forecast</VZT:Title></Title>
		<STYLE>
			.cur{mso-number-format:"\#\,\#\#0\.00";}
			.shade{background-color:inactivecaptiontext;}
			#cfTab tr{cursor:default;}
		</STYLE>
	</HEAD>
	<BODY>
		<SPAN class=box3>Month<br>
		<SELECT onchange=setMonth()>
		<VZT:Field Type="Script" Name="getOptions"></VZT:Field>
		</SELECT>
		</SPAN>
		<table class=colTable id=cfTab width=100% cellpadding=0
	cellspacing=0>
			<thead><tr>
				<td>Day of Month
				<td>Opening
				<td>Invoices
				<td>Bank Orders
				<td>Wages
				<td>Purchases
				<td>Other
				<td>Closing
			</thead>
	<VZT:Field Type="Script" Name="getDaysEntries"></VZT:Field>
		</TABLE>
	</BODY>
</HTML>
```

We also need a few styles in order to be able to choose the look of the fonts and colours etc.



Day of Month	Opening	Invoices	Overheads	Wages	Purchases	Other	Closing
01 Tuesday	182,133.26						182,133.26
02 Wednesday	182,133.26						182,133.26
03 Thursday	182,133.26				-114.66		182,018.60
04 Friday	182,018.60			-3,500.00	-1,143.28		177,375.33
05 Saturday	177,375.33				-988.35		176,386.98
06 Sunday	176,386.98	566.94					176,953.91
07 Monday	176,953.91		-1,590.50				175,363.41
08 Tuesday	175,363.41						175,363.41
09 Wednesday	175,363.41						175,363.41
10 Thursday	175,363.41						175,363.41
11 Friday	175,363.41			-3,500.00			171,863.41
12 Saturday	171,863.41						171,863.41
13 Sunday	171,863.41						171,863.41
14 Monday	171,863.41					-560.89	171,302.52
15 Tuesday	171,302.52						171,302.52
16 Wednesday	171,302.52						171,302.52
17 Thursday	171,302.52						171,302.52
18 Friday	171,302.52			-3,500.00			167,802.52
19 Saturday	167,802.52						167,802.52
20 Sunday	167,802.52						167,802.52
21 Monday	167,802.52						167,802.52
22 Tuesday	167,802.52						167,802.52
23 Wednesday	167,802.52						167,802.52
24 Thursday	167,802.52						167,802.52
25 Friday	167,802.52			-3,500.00			164,302.52
26 Saturday	164,302.52						164,302.52
27 Sunday	164,302.52						164,302.52
28 Monday	164,302.52		-150.85	-12,500.00			151,651.67
29 Tuesday	151,651.67						151,651.67
30 Wednesday	151,651.67						151,651.67
Totals	171,073.91	566.94	-1,741.35	-26,500.00	-2,246.28	-560.89	170,057.85

Script to control which month's information to display

In order to use the select element to control the display we need to implement:

1. client-side script to refresh the page when we change the display month
2. server-side script to display the select element appropriately

The client script gives us an insight into the way BCA navigates between pages in a stateful way.

```
<script language="jscript">
        function setMonth(){
                navigate("Report[1,,,CashFlow,,Cash Flow Forecast,," +
window.event.srcElement.value + "]");
        }
</script>
```

The NAVIGATE method is supplied by a supporting MainPage.js script module which submits a form which is always directed at the same default.asp system page.

The first parameter for this method is the active Page target (Report) followed by a square bracketed comma-separated list of NavParams which is accessible server-side. In this case the target is a generic report handler whose NavParams allow you to set the name of the report to call, its title and an OpenArgs parameter. We will use this last parameter to pass a date in the month of interest.

The main part of the server script is a function to build a set of option elements for the select element.

We check the OpenArgs parameter to determine which option is to be selected and set a global date parameter to be used by the function building the report table body.

Note the use of the MSTR class to perform efficient string concatenation.

```
<VZT:SCRIPT LANGUAGE="vbscript">
dim firstOfSelectedMonth 'global variable

function firstOfMonth(dt) 'this function returns the reverse first of month
        firstOfMonth = year(dt) & "/" & month(dt) & "/1"
end function
function mthCaption(dt) 'this function returns select option text
        mthCaption = monthname(month(dt)) &  " " & year(dt)
end function
function getOptions(vals,group) 'this function returns select options
        firstOfSelectedMonth = mparams("openargs")
        'if there is no date passed in set it to the first of this month
        if not isdate(firstOfSelectedMonth) then
                firstOfSelectedMonth = firstOfMonth(Date)
        end if
        for i = 0 to 6 'build the options list for 6 months from now
                optMonth = firstOfMonth(dateadd("m",i,Date))
                mstr.add "<option "
                if firstOfSelectedMonth = optMonth then mstr.add "selected "
                mstr.add "value='" & optMonth & "'>"
                mstr.add mthCaption(optMonth) & vbnewline
        next
        getOptions = mstr.text: mstr.clear
end function
</VZT:SCRIPT>
```

A sample output of the getOptions function is shown below:

```
<option selected value='2004/6/1'>June 2004
<option value='2004/7/1'>July 2004
<option value='2004/8/1'>August 2004
<option value='2004/9/1'>September 2004
<option value='2004/10/1'>October 2004
<option value='2004/11/1'>November 2004
<option value='2004/12/1'>December 2004
```

Script to draw the month-to-view calendar rows and add cash flow information

The script to build the table rows is more complex, and requires that we create a stored procedure to source the required information quickly and efficiently (see a subsequent section to see how we do this). This stored procedure will return two recordsets, the first simply with a single row of the most recent bank synchronization details and the second with all the transactions between then and the end of the month of interest. We will return all the rows for the second recordset into an array using the getRows method as this is most efficient.

We can evaluate the number of days in the month of interest using the native dateserial function with a daypart of 0, and use the weekday function to decide whether to shade the day as part of the weekend. Prior to stepping through and creating the days-of-the-month rows, we need to start stepping through the transactions data array until the beginning of the month to derive an opening balance.

Once into the dayCounter loop we create a colData array to ensure relevant transactions get added to the correct column, regardless of the order in which column data is returned from the recordset. This technique also allows us to choose what columns we want to show and bundle other transaction amounts into the OTHER column. We also use a totalizing colTotal array to provide a totals row under the month table. In the case of opening and closing balance columns we have chosen to calculate an average for these. Unfortunately we cannot easily use a join method on the array to provide final string output as we want to format the numbers nicely.

```
<VZT:SCRIPT LANGUAGE=vbscript>
dim firstOfSelectedMonth 'global variable

function getDaysEntries(vals,group)
dim colTotal(8, colData()
          for i = 0 to 8: colTotal(i) = 0: next'initialise array for column totals
          daysInMonth = Day(DateSerial(Year(firstOfSelectedMonth), Month(firstOfSelectedMonth)+1, 0)) 'get the days count
          set rs = mdoman.connection.execute("usp_FcastPayment_Get '" & firstOfSelectedMonth & "'",,0)
          syncBal = rs("SyncValue"):'get the first sp recordset which has syncvalue
          aTransData = rs.NextRecordset.GetRows'now get the second rs rows with the debit data
          lTransCount = ubound(aTransData,2)
          bal = syncBal: lTransRow = 0'find the months opening balance by adjusting balance since syncdate
          do while lTransRow < lTransCount and (datevalue(aTransData(0,lTransRow)) < datevalue(firstOfSelectedMonth))
                    bal = bal + aTransData(2,lTransRow): lTransRow = lTransRow + 1
          loop
          for dayCounter = 1 to daysInMonth
                    dayOfWeek = weekday(dateadd("d",dayCounter-1,firstOfSelectedMonth))
                    select case dayOfWeek'if weekend then shade the row tag
                              case 1,7: mstr.add "<tr class=shade>"
                              case else:          mstr.add "<tr>"
                    end select
                    'columns 0:dayOfMonth 1:dayOpen 2:dayInvoices 3:dayBankOrders 4:dayWages 5:dayPurchases 6:dayOther 7:dayClose
                    redim colData(7) 'setup /clear an array to hold the column data so its independent of ordering in the recordset
                    colData(0) = vbnewline & "<td>" & WeekdayName(dayOfWeek) & " " & dayCounter
                    colData(1) = bal: colTotal(1) = colTotal(1) + bal'set the days opening balance
                    'iterate through the transaction data for each day
                    do while day(aTransData(0,lTransRow)) = dayCounter
                              vTrans = aTransData(2,lTransRow): bal = bal + vTrans
                              select case aTransData(1,lTransRow) 'this is the category
                                        case "Sales Invoices":                    lColumn = 2
                                        case "Bank Orders":          lColumn = 3
                                        case "Purchase Orders":                   lColumn = 5
                                        case "Wages":                             lColumn = 4
                                        case else:                                          lColumn = 6
                              end select
                              colData(lColumn) = colData(lColumn) + vTrans'now add the transaction data to the correct cell
                              colTotal(lColumn) = colTotal(lColumn) + vTrans
                              if lTransRow = lTransCount exit do else lTransRow = lTransRow + 1
                    loop

                    colData(7) = bal: colTotal(7) = colTotal(7) + bal'completed the days transactions so set closing balance
                    mstr.add colData(0) 'join the array to produce the row output
                    for lColumn = 1 to 7
                              if colData(lColumn) <> 0 then
                                        mstr.add "<td align=right class=cur>": mstr.add formatnumber(colData(lColumn),2)
                              else
                                        mstr.add "<td> "
                              end if
                    next
          next
          mstr.add vbnewline & "<thead><tr><td>Totals" 'now we add the final totals row
          for lColumn = 1 to 7
                    if lColumn = 1 or lColumn = 7 then
                              mstr.add "<td align=right class=cur>"
                              mstr.add formatnumber(colTotal(lColumn)/daysInMonth,2)
                    else
                              mstr.add "<td align=right class=cur>"
                              mstr.add formatnumber(colTotal(lColumn),2)
                    end if
          next
          mstr.add "</thead>"
          getDaysEntries = mstr.text
end function
</VZT:SCRIPT>
```

A stored procedure to return relevant transaction information

For the purpose of building our cash-flow estimator, we are interested in returning information on all bank debits (and credits) which are expected to occur between the most recent BankSync date and the end of the month-of-interest.

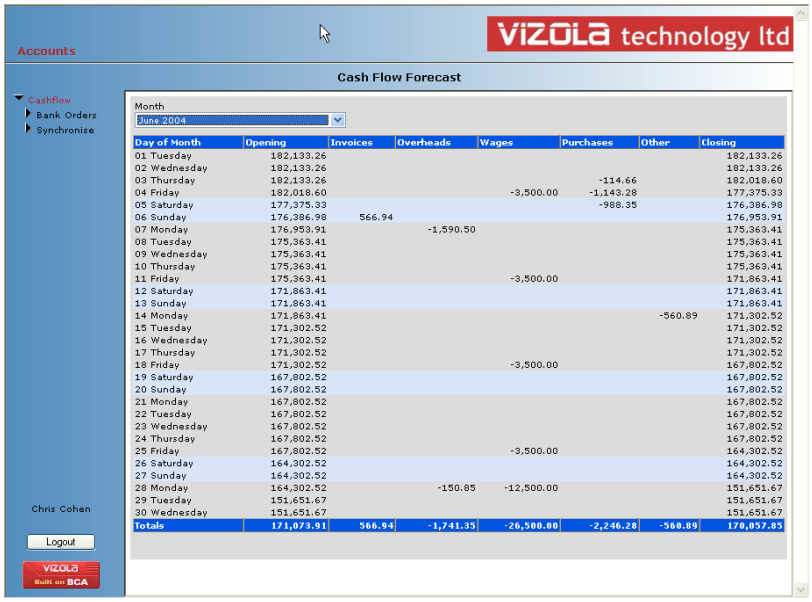
The most efficient way to do this is to create a SQL stored procedure which has one parameter passed in, being a date in the month-of-interest.

We are simply interested in summing the debits/credits on a daily basis for the period of interest.

```
CREATE      PROCEDURE dbo.usp_FcastPayment_Get

          @startdate smalldatetime = null --this is the first of the month-of-interest
-------we need to get the info from the last bank sync and return all adjustments from then
-------until the end of the month of interest
AS

declare @syncdate smalldatetime
declare @syncvalue money
declare @enddate smalldatetime

-------this gets the last day of the month-of-interest for where clause
-------being this month if nothing is passed in
if @startdate = null select @startdate = cast(getdate() as int)
set @startdate = dateadd(d,-day(@startdate)+1,@startdate)
set @enddate = dateadd(m,1,@startdate)-1

-------this gets the latest banksync information using BCA fns to extract
-------date and money types from the text type catval columns
select    top 1 @syncdate = dbo.fn_getcatentrydate(catval,1), @syncvalue =
dbo.fn_getcatentrymoney(catval,2)
from tbl_doCatValue c inner join tbl_doDef d on c.DefID=d.ID
where catname='default' and d.name='banksync'
order by cast(catval as varchar) desc

------this select returns the banksync info in the first recordset
select @syncdate as SyncDate, @syncvalue as SyncValue

------this second select returns the bank debits between @syndate and @enddate
select Expected, Category, Sum(Amount) as Amount
from (
------these are the debits / credits setup as bank orders
select    dbo.fn_getcatentrydate(catval,1) as Expected,
dbo.fn_getcatentry(catval,3) as Category, -sum(dbo.fn_getcatentrymoney(catval,2)) as Amount
from tbl_docatvalue where catname='debits' and dbo.fn_getcatentrydate(catval,1) between @syncdate and
@enddate
group by dbo.fn_getcatentrydate(catval,1), dbo.fn_getcatentry(catval,3)
------these are the expected payments from customers from invoicing
union select dbo.fn_getcatentrydate(catval,1), 'Sales Invoices', sum(dbo.fn_getcatentrymoney(catval,2))
from tbl_docatvalue where catname='topay' and dbo.fn_getcatentrydate(catval,1) between @syncdate and
@enddate
group by dbo.fn_getcatentrydate(catval,1)
------these are the payments to be made to suppliers for purchase orders
union select dbo.fn_getcatentrydate(catval,1), 'Purchase Orders', -
sum(dbo.fn_getcatentrymoney(catval,2))
from tbl_docatvalue where catname='payable' and dbo.fn_getcatentrydate(catval,1) between @syncdate and
@enddate
group by dbo.fn_getcatentrydate(catval,1)
) tmp
group by Expected, Category
order by Expected, Category
```

The final output plus further enhancements

Our final cash-flow estimator output is shown here. Included also are columns resulting from expected cash flows from purchases and invoicing (which is outside the scope of this project).



Day of Month	Opening	Invoices	Overheads	Wages	Purchases	Other	Closing
01 Tuesday	182,133.26						182,133.26
02 Wednesday	182,133.26						182,133.26
03 Thursday	182,133.26				-114.66		182,018.60
04 Friday	182,018.60			-3,500.00	-1,143.28		177,375.33
05 Saturday	177,375.33				-988.35		176,386.98
06 Sunday	176,386.98	566.94					176,953.91
07 Monday	176,953.91		-1,590.50				175,363.41
08 Tuesday	175,363.41						175,363.41
09 Wednesday	175,363.41						175,363.41
10 Thursday	175,363.41						175,363.41
11 Friday	175,363.41			-3,500.00			171,863.41
12 Saturday	171,863.41						171,863.41
13 Sunday	171,863.41						171,863.41
14 Monday	171,863.41					-560.89	171,302.52
15 Tuesday	171,302.52						171,302.52
16 Wednesday	171,302.52						171,302.52
17 Thursday	171,302.52						171,302.52
18 Friday	171,302.52			-3,500.00			167,802.52
19 Saturday	167,802.52						167,802.52
20 Sunday	167,802.52						167,802.52
21 Monday	167,802.52						167,802.52
22 Tuesday	167,802.52						167,802.52
23 Wednesday	167,802.52						167,802.52
24 Thursday	167,802.52						167,802.52
25 Friday	167,802.52			-3,500.00			164,302.52
26 Saturday	164,302.52						164,302.52
27 Sunday	164,302.52						164,302.52
28 Monday	164,302.52		-150.85	-12,500.00			151,651.67
29 Tuesday	151,651.67						151,651.67
30 Wednesday	151,651.67						151,651.67
Totals	171,073.91	566.94	-1,741.35	-26,500.00	-2,246.28	-560.89	170,057.85

There are countless other enhancements which could easily have been incorporated, including:

1. create custom editor for BankOrders supporting dropdown selects instead of text fields for STATUS and CATEGORY
2. adding a caption property to the BankDebits which appears as a tooltip on mouseover
3. adding a button to export to Excel
4. adding a button to list transaction detail for the month
5. adding a hyperlink on the dates to popup a window to synchronise your bank balance or add further bank orders
6. suppressing balance data when unchanging

About Vizola's Business Component Architecture

The BCA system consists of:

1. An object-oriented database built on SQL Server
2. Object Definition Editor which enables:
    a. Managing object definitions and object instances
    b. Creating and managing menu hierarchies
    c. Managing form field validation using regular expressions
    d. Creating HTML reports for use with BCA catalogues and objects
3. Capable and extensible default object handlers for pages and reports
4. Extensive and growing library of custom object handlers

During the course of constructing this project we have seen something of BCA and how it works. Hopefully it illustrates to developers and interested parties something of the benefits to be gained by using BCA as a foundation for their own applications. Licenses are available to use the system, complete with source code. Please contact Chris Cohen for further information.