Writing ActiveX Controls for use in Synthesys

Overview
This document is aimed at developers wanting to know how to write ActiveX controls to extend the functionality of Synthesys.

Document History
This document replaces and consolidates information previously found in 'Synthesys for Systems Integrators' document.

First version	Luke Harris, 12th August 2003
Updated with note on VB controls	Luke Harris, 26th July 2006
Update for Synthesys 3.2	Luke Harris, 2 October 2007
Appended appendix with information about branding	Oleg Kashayev, 24th November 2008
Reviewed for Synthesys v4.3	Luke Harris, 3rd January 2013

Email : Luke.Harris@noetica.com

Introduction
There are no special requirements for writing ActiveX controls for Synthesys. We have successfully in the past used 3rd party controls which were originally written for Visual Basic for example.

The companion document 'Synthesys for Systems Integrators' lists some examples of ActiveX controls that Noetica has written for specific requirements, and indicates some other ways of integrating to Synthesys.

Noetica has a wide experience of writing ActiveX controls for different purposes in Synthesys and so can provide advice on the best techniques. Please contact us if you need any extra help.

Deploying ActiveX controls
Synthesys can aid in distributing ActiveX controls to client workstations. In principle all you need to do is put any files (including the OCX or DLL for the control) into the synthesys\bin directory on the server; Synthesys will then automatically copy new versions of files to workstations and register them on startup.

To make the ActiveX control available in the Callflow designer, it needs to be put into the Synthesys gallery. When inserting an ActiveX into the gallery you can designate which fields should be saved into the database (DB properties) and how wide they need to be (strings are stored as VARCHAR data types). Properties can alternatively be marked as PUBLIC, which means they are available for calculations made in scripting, but won't be persisted in the database.

Developing ActiveX controls
At Noetica we generally use Microsoft Visual C++ to create ActiveX components. Synthesys v2.5 uses Microsoft Visual C++ v6.0, Synthesys v3.2 uses the C++ compiler with Visual Studio.NET 2005.

Other programming environments are also acceptable, but be careful if the resulting ActiveX controls need specific deployment options such as registry settings or third

party DLL files. Synthesys already installed the DLLs required for applications created using the appropriate version of Visual Studio.

Testing in the initial stages can be done outside Synthesys (for example using Microsoft's TSTCON32.exe program). During later stages of testing the ActiveX should be checked within Synthesys to make sure it behaves as expected.

Examples of controls
Some of these are relatively trivial, some have taken months of development work.

- Extracting data from an Enterprise database.
- Screen scraping a Mainframe order processing system to create, modify and view order, customer and shipping details.
- A control for a Video Conferencing company, allowing studios in different parts of the world to be booked, and fees to be calculated depending on time of day, number of studios booked.
- A control for a newspaper company, which would find the appropriate local newsagent for delivering the newspaper, using a variety of algorithms depending on circumstance.
- A control which can calculate appropriate sales discounts.
- A control which does 'auto correct' style spell checking, allowing replacements of common abbreviations to speed agent typing.
- A 'clipboard' control, which can be used to communicate data with external programs (for example CTI interfaces).
- A ticket control, which is used to allocate tickets for theatres and shows.

Useful Techniques
If the ActiveX is required to communicate to an enterprise system using some API, it is often convenient to do this in a client/server architecture, so that it is not necessary to deploy extra applications on agent workstations. So for example the ActiveX would make a request to a server application (over TCP/IP or SOAP for example) to search the Enterprise database for a record, and would receive the response for display to the operator. Calls should be asynchronous to preserve user interface responsiveness.

If you require agents to wait (for response from the server for example) before moving to the next section, you can use make one of fields of the ActiveX mandatory, so that the 'Take Calls' module won't progress until it is set. Alternatively, the ActiveX can implement a method called 'Validate' which will do custom validation, returning TRUE(1) if it is OK to continue or FALSE(0) otherwise.

If a call is parked or held, the ActiveX will be recreated when reloaded based on all the values of its properties. So all state that needs to be saved and restored must be available in properties.

If the ActiveX can implements a 'Maintenance' method, then this can be invoked from the Callflow editor to customise the control by for example showing a dialog box or even running an external application. This allows for additional flexibility when a standard 'Property page' isn't appropriate.

Synthesys implements the standard ambient properties, including DISPID_AMBIENT_USERMODE to indicate if the host is the 'Take Calls' module or the Webflow designer. Synthesys also implements the following ambient properties which can be useful in certain circumstances.

Name	***DISPID***	***Description***
DISPID_AMBIENT_CAMPAIGN	5000	Campaign prefix, such as

		AAA01
DISPID_AMBIENT_SECTION	5001	Section Name.
DISPID_AMBIENT_CONTROL	5002	Control Name.
DISID_AMBIENT_ DATASOURCENAME	5003	ODBC data source name; Phoneyx in design mode, phoenix in release mode.
DISPID_AMBIENT_SEQUENCE_ID	5005	Sequence ID (Run Mode)
DISPID_AMBIENT_CUSTOMER_ID	5006	Current customer ID (Run Mode)
DISPID_AMBIENT_USERID	5007	ID of logged on user.

Controls developed using Visual Basic
If you are developing controls using Visual Basic, there are two points to be aware of. First is that you must ensure that the same ID is used for the control each time you build it; by default Visual Basic picks a new GUID each time. You can ensure the same GUID is used by selecting the 'binary compatibility' option in the project settings.
Secondly you will need to ensure that any Microsoft DLL files that the ActiveX needs are shipped to all workstations needing them (msvbvm60.dll for example).

Branding

```
To support Synthesys branding Activex control should read the
following standard ambient properties:

DISPID_AMBIENT_BACKCOLOR
DISPID_AMBIENT_FORECOLOR
DISPID_AMBIENT_FONT

To store this ambient information it recommended to declare the
following members in your activex control class:

class MyCtrl : public COleControl
{
 ....
protected:
 CFont m_Font;
 CFontHolder m_ControlFont;

 COLORREF m_clrBack;
 COLORREF m_clrFore;
 CBrush   m_bkBrush;

 ....
};

Overwrite COleControl::OnAmbientPropertyChange method to react on
ambient property changes.

Note that if it's necessary for system to set more than one ambient
property at a moment this method will be called with parameter
DISPID_UNKNOWN.

Note that if ambient font property is changed it is required to
recalculate ActiveX control elements sizes, readjust their positions
and possibly change the size of control itself to fit new sizes of
contained elements. So we call ResizeControlElements method.
```

```
void MyCtrl::OnAmbientPropertyChange(DISPID dispid)
{
 long nColour;

 if (dispid == DISPID_AMBIENT_BACKCOLOR || dispid == DISPID_UNKNOWN)
 {
  BOOL res =
GetAmbientProperty(DISPID_AMBIENT_BACKCOLOR,VT_I4,&nColour);
  m_clrBack = res ? TranslateColor(nColour) :
GetSysColor(COLOR_BTNFACE);

  m_bkBrush.DeleteObject();
  m_bkBrush.CreateSolidBrush(m_clrBack);
 }

 if (dispid == DISPID_AMBIENT_FORECOLOR || dispid == DISPID_UNKNOWN)
 {
  res = GetAmbientProperty(DISPID_AMBIENT_FORECOLOR,VT_I4,&nColour);
  m_clrFore = res ? TranslateColor(nColour) :
GetSysColor(COLOR_WINDOWTEXT);
 }

 if (dispid == DISPID_AMBIENT_FONT || dispid == DISPID_UNKNOWN)
 {

  LPFONTDISP  pFontDisp = AmbientFont();
  if(pFontDisp)
  {
   m_ControlFont.InitializeFont( &_fontdescControl, pFontDisp);
   pFontDisp->Release();

   if(m_ControlFont.m_pFont)
   {
    HFONT hFont = 0;
    HRESULT hres = m_ControlFont.m_pFont->get_hFont(&hFont);

    if (hres == S_OK)
    {
     LOGFONT lf;
     CFont::FromHandle(hFont)->GetLogFont(&lf);
     m_Font.CreateFontIndirect(&lf);
    }
   }

   SetFont(pFontDisp);
  }

  if(m_Font.m_hObject==NULL)
   m_Font.CreatePointFont(CONTROL_FONTSIZE,CONTROL_FONTNAME);

  ResizeControlElements();
 }

 COleControl::OnAmbientPropertyChange(dispid);
}
```

First thing in ResizeControlElements method we should specify new ambient font value for each activex control element by calling their SetFont methods. Then we access window CDC object and select our new ambient font value into it. Now we can use GetTextExtent and GetTextMetrics methods to calculate the sizes of control elements.

```
void MyCtrl::ResizeControlElements()
{
 if  (!::IsWindow(m_hWnd))
  return;

 MyCtrlElement1->SetFont(&m_Font);
 ......

 CDC* pDC = GetDC();
 if  (pDC == NULL)
  return;

 CFont* pOldFont = pDC->SelectObject((CFont *)&m_Font);

 TEXTMETRIC tm;
 pDC->GetTextMetrics(&tm);

 ....

}
```

We can call OnAmbientPropertyChange with parameter DISPID_UNKNOWN in OnCreate method to read ambient properties and adjust control elements sizes and positions for the first time. Control elements must be created before calling OnAmbientPropertyChange method.

```
int MyCtrl::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
 if  (COleControl::OnCreate(lpCreateStruct) == -1)
  return -1;

 //Create your control elements here
 ....

 OnAmbientPropertyChange(DISPID_UNKNOWN);

 return 0;
}
```

To make control elements use back and fore colours provided by ambient properties it necessary to handle WM_CTLCOLOR message.

```
BEGIN_MESSAGE_MAP(MyCtrl, COleControl)
 .....
 ON_WM_CTLCOLOR()
END_MESSAGE_MAP()


HBRUSH MyCtrl::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
 HBRUSH hbr = COleControl::OnCtlColor(pDC, pWnd, nCtlColor);
 if (nCtlColor == CTLCOLOR_STATIC
  || nCtlColor == CTLCOLOR_BTN
  || nCtlColor == CTLCOLOR_DLG
  || nCtlColor == CTLCOLOR_SCROLLBAR)
 {
  pDC->SetTextColor(m_clrFore);
  pDC->SetBkColor(m_clrBack);
  return m_bkBrush;
```

```
 }
 else
  pDC->SetTextColor(m_clrFore);

 return hbr;
}
```

Note that in OnCtlColor handler we path ambient backcolour only for those control elements which background supposed to be same as the one of containing them control. For example if we provided ambient backcolor for CEdit control, then it internal space will be say grey instead of usual white.

In activex control OnDraw procedure it's recomented to call RedrawWindow function for each control element to be sure they are properly drawn over section background fill

```
void MyCtrl::OnDraw(CDC* pdc, const CRect& rcBounds, const CRect&
rcInvalid)
{
 pdc->FillRect(rcBounds, &m_bkBrush);

      ::RedrawWindow(MyCtrlElement1->hWnd, NULL, NULL,
      RDW_INVALIDATE|RDW_UPDATENOW|RDW_ERASE|RDW_INTERNALPAINT|RDW_FR
      AME);

 .....
}
```