



# **TriSys 6.01**

## **Scripting Engine Bible & Tutorial**

*TriSys provides very flexible customisability by way of form design, config fields, reports, actions and stored procedures to implement business logic. From 2000, TriSys provided a scripting engine to allow scripts to be written so that the application can be programmed by non-professional programmers during implementation.*

Prepared by:

**Garry Lowther**  
**TriSys Business Software Ltd**

Original Draft: September 2000

Current Version: Wednesday, 27 October 2004

## Table of Contents

1. Introduction .....	4
2. Event Notes.....	5
3. Virtual Entities/Forms.....	6
4. TriSys Object Model.....	8
5. Sample TriSys VB Script .....	10
6. Example of setting ellipses lookups .....	16
7. Step by Step Example of Master-Detail Dynamic Grid Handling.....	17
8. Field Visibility Example Code – How to Use FormFields within ShowForm.....	22
9. Button Visibility Example Code – How to Use FormButton within ShowForm .....	24
10. Setting Selected Tab In Script .....	25
11. Detecting Which Button is Pressed .....	26
12. Firing an Action Button .....	27
13. Firing an Action From Script Code .....	28
14. Validating Action Values.....	30
15. Processing Requirement Shortlist .....	32
16. Creating a Lookup Combo.....	34
17. Asking the User to Choose from a List .....	35
18. Using SQL Selection Object.....	36
19. Modal Virtual Form Interaction .....	37
20. Action Form Collection Manipulation .....	40
21. Firing A Report From Script.....	41
22. Using Classes in VBScript – Object Oriented Programming .....	42
23. Programming Grid Column/Cell Colours .....	43
24. Multi-Select Grid Rows.....	45
25. Launching Core Entity Forms From Script.....	46
26. Using FileSystemObject in VBScript .....	48
27. Read all Grid Rows.....	49
28. External Document Launching.....	50
29. Global Parameter Value.....	51
30. Launching E-Mail Client.....	52
31. Colouring Tabs.....	55
32. Crystal Report Form Script.....	56
33. Automating Word.....	58
34. Searching for Requirements within Post Code Radius .....	59
35. Updateable Grid for Data Entry .....	60
36. Programmed Fields Re-Appearing on Record Save .....	62
37. AddItem Grid Rows .....	63
38. Unbound Virtual Grid Population .....	64
39. Refreshing Grids on Other Entity Forms .....	65

40.	Running Mail Merges From Script .....	66
41.	Diagnostics and User Activity Logging for Debugging and Audit Trails.....	67
42.	Closing Action Forms .....	68
43.	Intercepting Shortlisting Events.....	69
44.	Post Code Lookup.....	70
45.	Using Bookmarks in Grids.....	71
46.	Accessing Grid SQL Query Recordset and Cloning.....	72
47.	Unbound Grids - Formatting Columns.....	73
48.	Fast Contact Search – Script Access .....	74
49.	Form_Unload – Preventing form from closing from Script.....	75
50.	Spin Increments – Setting Visibility from Script .....	76
51.	FreeTextSearchEngine – Searching CV’s using ISYS from Script .....	77
52.	Access System Settings .....	78
53.	Appendix A - Type Mismatch Errors .....	79

## 1. Introduction

This shows which events are raised for each object:

FormObjectEvent_AfterInitialise	' Grid
FormObjectEvent_AfterInsert	' Grid
FormObjectEvent_AfterLoad	' Form
FormObjectEvent_AfterLoadRecord	' Form
FormObjectEvent_AfterSaveRecord	' Form
FormObjectEvent_AfterShortlist	' Form (Candidate Search, Requirement Only)
FormObjectEvent_AfterUpdate	' Grid
FormObjectEvent_BeforePopulation	' Grid
FormObjectEvent_BeforeSaveRecord	' Form
FormObjectEvent_Change	' Field
FormObjectEvent_Click	' Button, Field, Menu, Grid, Tab
FormObjectEvent_DblClick	' Field, Grid
FormObjectEvent_DisplayCell	' Grid
FormObjectEvent_ExternalRefresh	' Form
FormObjectEvent_GotFocus	' Field, Grid
FormObjectEvent_KeyPress	' Field, Grid
FormObjectEvent_LostFocus	' Field, Grid
FormObjectEvent_RowLoaded	' Grid
FormObjectEvent_RowSelection	' Grid
FormObjectEvent_Unload	' Form

## 2. Event Notes

This section contains further information about events as they are enhanced.

*Public Sub Form\_ExternalRefresh (FormName, Id)*

*End Sub*

This event should be captured every time a record in the application is saved.

It uses the same mechanism as the internal source code which receives an event when a record is updated so that any interested fields/grids etc can be refreshed to show the updated information.

For example if a company address changes, any forms showing the old company address need to show the new one.

Script can take advantage of this mechanism by capturing this event and determining whether the saved form and record has any cause for the script to take action.

An example might be when a script is controlling master/detail grids and the user drills down into the detail record. Both grids will need to be refreshed if the user updates the record.

This mechanism provided the necessary event.

*Note: If an entity form contains a grid from another entity and the user drills down into this, TriSys will automatically refresh the first form after the second has been updated. This requires no scripting.*

*Public Sub Grid\_BeforePopulation (SQL, Cancel)*

*End Sub*

This event is called just before a grid is populated. The SQL parameter contains the SQL derived dynamically from both the form designed query and the on-screen widgets which have been manipulated by the user.

It gives the script developer the chance to override the SQL by for example, parsing in extra search parameters which have been created using script code.

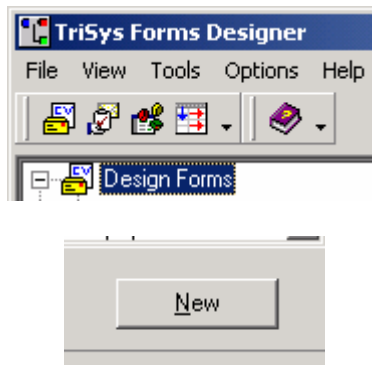
The Cancel parameter can be overridden by the script using the *FunctionStatus* object in order to cancel the form from processing the SQL if the script wishes to manipulate the grid entirely autonomously from the form logic. This technique provides absolute script control to override the normal search behaviour.

### 3. Virtual Entities/Forms

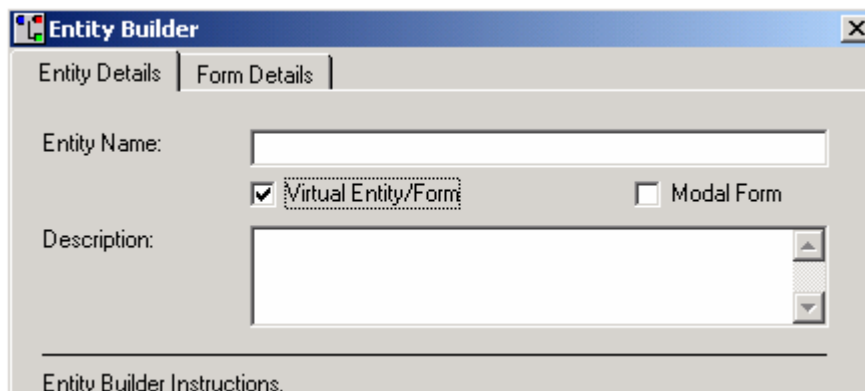
Form Designer has been extended (31<sup>st</sup> March 2001) to allow for 'virtual entities'. These are defined as ad-hoc forms with no underlying entity. All entity fields would be populated entirely by the scripting object model.

The form can be invoked from anywhere and form designer will allow buttons, fields etc to be added to it with script code behind.

Virtual entities/forms are created in exactly the same manner as normal entities i.e. using the entity builder:



Pressing the New button loads the Entity Builder:

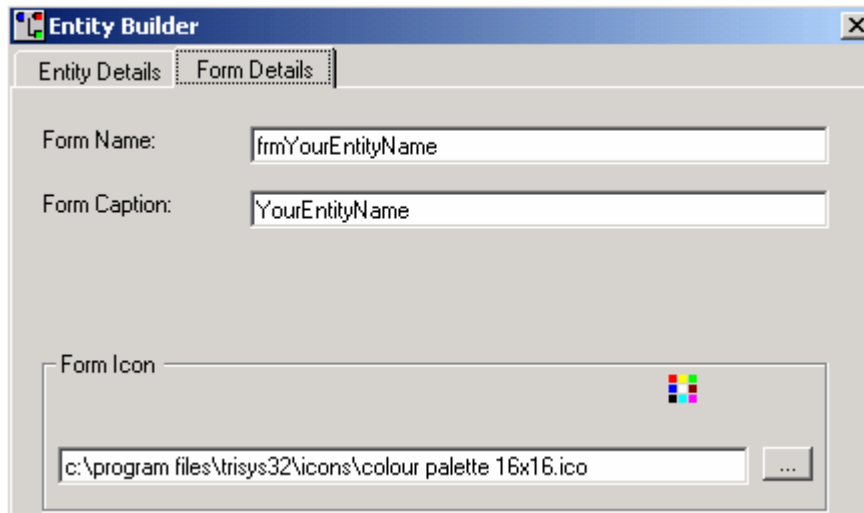


Note that when the Virtual Entity/Form check box is ticked, the Modal Form check box appears, and the Contact Management Links and Entity Links tabs become invisible (they are not of any use).

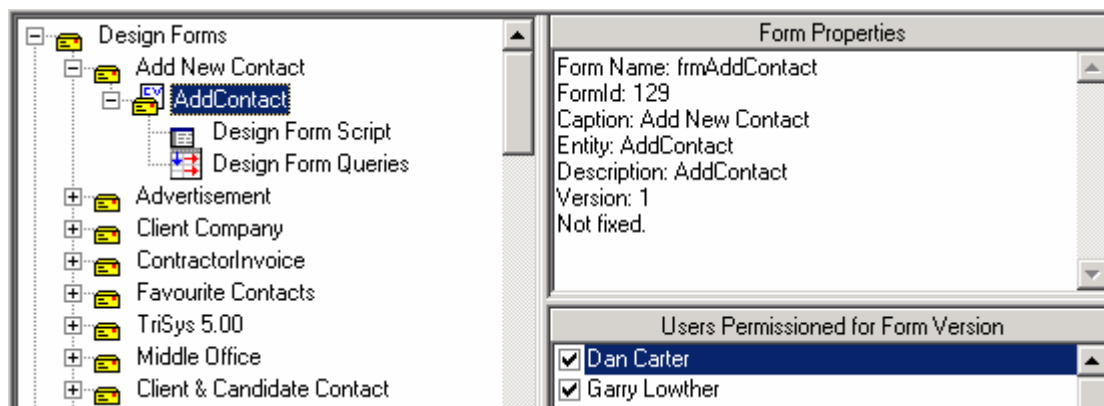
Enter the name of your entity and provide a sensible description to help self-document the entity.

If you want the form to be loaded modally, i.e. shown on-top of all other forms, centred on the screen, and has all focus such that the user must complete the form to return to TriSys, then check the Modal Form check box.

Select the Form Details tab. This allows you to set a caption for the form, and to assign an icon for it which will allow it to be available from the main list bar and it will also be shown on the top left of the form when loaded:



When you save your entity, you will then design the form in the same fashion as other forms, actions and queries.



The main difference is that there are no fixed fields or reference fields and no default buttons. This is because this is a virtual form i.e. it sits within the TriSys infrastructure but does not have any business logic to read or save from the TriSys object model. The whole idea is that you cde this logic entirely through script. This has the main advantage of letting you write your own application rules without having to tweak existing rules.

When you create config fields in the form designer for virtual forms, no fields are created on the SQL Server database. It is the responsibility of the script developer to write code which will read/write these fields to/from whatever data source is necessary, which of course may not be a SQL Server database.

When developing script, the standard objects and mechanisms are identical to writing script for other forms, such that virtual form code looks identical to other forms. You can cut & paste code from all parts of the system and re-use global functions etc...

The method of accessing form fields is the same too, with the table name being the name of the newly created entity + "ConfigFields" which is in keeping with the standard system nomenclature for accessing config fields e.g.:

```
With FormObjectsLocal.ShowForm
    sForenames = .GetFieldValue("AddContactConfigFields", "Forenames")
    sSurname = .GetFieldValue("AddContactConfigFields", "Surname")
End With
```

## 4. TriSys Object Model

This section contains detailed documentation of the TriSys Object Model. This object model is split into 2 parts for the purposes of the scripting engine:

- Forms Objects – these provide access to the form upon which the scripts are invoked
- Global Objects – these provide access to the global business object model

Form Objects Examples:

```
Dim formObjectsfrmMain, formField, formObj
Set formField = FormObjectsLocal.GetFormObjectByName("Contact", "Male")
If formField.formWidget.Value = 1 then
    formField.formWidget.Text = "Man"
Else
    formField.formWidget.Text = "Woman"
End if

' Access another form using different global collection
Set formObjectsMain = FormObjectsGlobal("frmMain")
If not formObjectsMain is nothing then
    Set formRef = formObjectsMain.FormRef
    formRef.Caption = "Someone changed sex!"
End if
```

The exposed TriSys Objects are:

<b>Global Object</b>	<b>Description</b>
<b>Initialisation</b>	UsrCfgUsers table fields for current user + collections of contacts, companies etc..
<b>TriSysDatabase</b>	The CDatabase object which exposes methods to access the SQL Server database.
<b>WordProcessor</b>	Invoke Word automation
<b>StringFunctions</b>	String manipulation functions
<b>MessageBox</b>	Show messages and ask questions of user
<b>FileSystem</b>	Access the underlying windows file system
<b>Selection</b>	Select business data e.g. contacts, companies from pop-up dialogue grids
<b>DataWidgets</b>	The grid functions to operate upon on-screen grids
<b>DbGet</b>	Get standard database functions
<b>Numbers</b>	General purpose number manipulation
<b>CrystalReports</b>	Underlying crystal report related functions not CRPE
<b>FormFunctions</b>	The functions which control TriSys forms
<b>Diagnostics</b>	Diags.out file access to record own messages to output file
<b>Miscellaneous</b>	Functions such as auto-dial, load web browser etc..
<b>ShellExecute</b>	Execute shell functions such as launch e-mail dialogue etc..
<b>Registry</b>	Access the registry
<b>AnimateWait</b>	Show animation dialogue similar to TriSys start up
<b>ProgressWait</b>	Show progress gauge or tick stages
<b>BigInput</b>	Capture any text input from a dialogue box
<b>ErrorHandling</b>	Standard error handler functions
<b>MailMerge</b>	Access to standard mail merge templates utilising Universal Data Source.tmm
<b>PostCodeLookup</b>	Access to the post code lookup object
<b>EMailGateway</b>	The e-mail client automation object for interacting with Outlook/Notes etc..
<b>FreeTextSearchEngine</b>	Access the ISYS free text search engine to search CV's

## 5. Sample TriSys VB Script

This section contains sample VB Script to demonstrate various features of the TriSys Object Model and VB Script.

```
' Contact form script.
' Contains business logic to deal with configurable form.

.....
' Globals to preserve state between invocations

Dim GotFocusColour      ' Colour of field normally - used in LostFocus/GotFocus
.....

Sub Button_Click
    MsgBox.AppMsg "Script: You pressed button: " &
FormObjectsLocal.EventControl.Caption
End Sub

.....
' Capture field click and synchronise the Title and Sex options
Sub Field_Click
    Set fieldObj = FormObjectsLocal.EventControl
    Set FieldDescription =
FormObjectsLocal.ShowForm.GetFieldDescriptionFromWidgetHwnd( fieldObj.hWnd )

' Turn on for debugging
'MessageBox.AppMsg "Field_Click: " & FieldDescription.TableName & "." &
FieldDescription.TableFieldName

    if Not FieldDescription is nothing then
        Select Case FieldDescription.TableFieldName
            case "ContactTitle"
                ProcessComboTitle fieldObj.Text

                case "ContactTypeClient", "ContactTypeCandidate",
"ContactTypeClientCandidate"
                    ProcessContactType

                case "CandidateStatus"
                    ProcessComboStatus fieldObj
        End Select
    end if
End Sub

.....
' Called when title combo is changed.
' Set the Sex accordingly
Sub ProcessComboTitle ( sTitle )
    select case sTitle
        case "Mr", "De heer", "Herr", "Monsieur"
            bMale = True
        case else
            bMale = False
    end select

    REM MsgBox.AppMsg "ProcessComboTitle: " & sTitle & ", Male=" & bMale

    ' Set in correct Check box after change
    set maleObj = FormObjectsLocal.ShowForm.GetFieldWidget("Contact",
"ContactMale")
    set femaleObj = FormObjectsLocal.ShowForm.GetFieldWidget("Contact",
"ContactFemale")
```

```

if not maleObj is nothing then maleObj.Value = Abs(bMale)
if not femaleObj is nothing then femaleObj.Value = Abs(Not bMale)

' Marital Status
set marriedObj = FormObjectsLocal.ShowForm.GetFieldWidget("Contact",
"ContactMarried")
if not marriedObj is nothing then
    if sTitle = "Mrs" then marriedObj.Value = 1
end if
End Sub

.....
' When Active status = Do not Use, then turn RED for danger
Sub ProcessComboStatus ( objStatus )

    if not objStatus is nothing then
        if objStatus.Text = "Do Not Use" then
            objStatus.BackColor = vbRed
        else
            objStatus.BackColor = vbYellow
        end if
    end if
End Sub

.....
'
' Called when contact type is changed.
' Hide/Show controls accordingly
Sub ProcessContactType ( )

    bClient = Not FormObjectsLocal.ShowForm.GetFieldValue("Contact",
"ContactTypeCandidate")

    set objAvailabilityDate = FormObjectsLocal.ShowForm.GetFieldWidget("Contact",
"AvailabilityDate")
    if not objAvailabilityDate is nothing then objAvailabilityDate.Visible = not
bClient

    set objAvailabilityDateLabel =
FormObjectsLocal.ShowForm.GetFieldWidget("Contact", "AvailabilityDate", 1)
if not objAvailabilityDateLabel is nothing then
    ' Only set the visibility if this button is in view
    If FormObjectsLocal.ShowForm.isWidgetVisibleOnCurrentTab( _
        objAvailabilityDateLabel.hWnd) Then
        objAvailabilityDateLabel.Visible = not bClient
    end if

    ' Set the name colours to those for each contact

    set objForenames = FormObjectsLocal.ShowForm.GetFieldWidget("Contact",
"Forenames")
    set objSurname = FormObjectsLocal.ShowForm.GetFieldWidget("Contact",
"Surname")

    if not objForenames is nothing and not objSurname is nothing then
        if bClient then
            with objForenames
                .BackColor = vbYellow
                .ForeColor = vbBlue
            end with
            with objSurname
                .BackColor = vbYellow
                .ForeColor = vbBlue
            end with
        end if
    end if
end Sub

```

```

        end with
    else
        with objForenames
            .BackColor = vbRed
            .ForeColor = vbYellow
        end with
        with objSurname
            .BackColor = vbRed
            .ForeColor = vbYellow
        end with
    end if
end if

exit sub

' Set in correct Check box after change
if bClient then
    FormObjectsLocal.FormRef.Tabby.Tabs(3).Caption = "Foreign Language"
else
    FormObjectsLocal.FormRef.Tabby.Tabs(3).Caption = ""
end if

End Sub

.....

Sub Field_GotFocus
    Set fieldObj = FormObjectsLocal.EventControl
    GotFocusColour = fieldObj.Backcolor

    fieldObj.Backcolor = vbCyan    'glbFieldFocusColour
End Sub

.....

Sub Field_LostFocus
    Set fieldObj = FormObjectsLocal.EventControl
    fieldObj.Backcolor = GotFocusColour

    ' Remember to call fields which are validated and affect colour afterwards!!!
    call Field_Click
End Sub

.....

Sub Field_KeyPress(KeyAscii)
    Set fieldObj = FormObjectsLocal.EventControl
    Set FieldDescription =
FormObjectsLocal.ShowForm.GetFieldDescriptionFromWidgetHwnd( fieldObj.hWnd )

' Turn on for debugging
'FormObjectsLocal.FormRef.Caption = "Field_KeyPress: " & Now & ": " & _
'     FieldDescription.TableName & "." & FieldDescription.TableFieldName

    if Not FieldDescription is nothing then
        Select Case FieldDescription.TableFieldName
            case "AlternativePostCode"
                KeyAscii = Asc(UCase(Chr(KeyAscii)))    ' Convert to
uppercase
        End Select
    end if

    ' Set function parameter - caller can retrieve that this has changed
    FunctionStatus.SetParameter("KeyAscii") = KeyAscii
End Sub

```



```

.....
' Company Script to add a button and respond to its events
' to change the colours of the screens etc..
,
' Oct 2000
' Garry Lowther
.....
dim m_ColIndex

Sub Form_AfterLoad
    Dim cmdObject

    Set cmdObject = _
        formObjectsLocal.LoadControl("VB.CommandButton", "btn")

    with cmdObject
        .Visible = True
        .Caption = "Press Me"
        .Tag = "THEONE"
        .top = 1000
        .left = 1000
        .ZOrder
    end with

End Sub

Sub Button_Click
    dim colour, i

    if FormObjectsLocal.EventControl.Tag = "THEONE" then
        'MessageBox.AppMsg "Script: You pressed button: " & _
        '    FormObjectsLocal.EventControl.Caption

        m_ColIndex = m_ColIndex + 1
        select case m_ColIndex
            case 1: colour = vbBlue
            case 2: colour = vbRed
            case 3: colour = vbGreen
            case 4: colour = vbYellow
            case 5: colour = vbCyan
            case 6: colour = vbMagenta
            case 7
                colour = vbWhite
                m_ColIndex = 0
        end select

        ' Colour controls to match
        formObjectsLocal.formRef.tabby.BackColor = colour
        formObjectsLocal.formRef.ctlFormCaptionController.GetToolbar.BackColor
= colour
        formObjectsLocal.formRef.scroller.BackColor = colour
        FormObjectsLocal.EventControl.Caption = "Press Me (" & m_ColIndex & ")"

        for i = 1 to formObjectsLocal.formRef.controls("txt").Count - 1
            formObjectsLocal.formRef.controls("txt")(i).BackColor = colour
        next
        for i = 1 to formObjectsLocal.formRef.controls("btn").Count - 1
            formObjectsLocal.formRef.controls("btn")(i).BackColor = colour
        next
        for i = 1 to formObjectsLocal.formRef.controls("pnl").Count - 1
            formObjectsLocal.formRef.controls("pnl")(i).BackColor = colour
        next
        for i = 1 to formObjectsLocal.formRef.controls("frame").Count - 1
            formObjectsLocal.formRef.controls("frame")(i).BackColor = colour
    end if
end Sub

```

```
        next
    end if
End Sub
```

## 6. Example of setting ellipses lookups

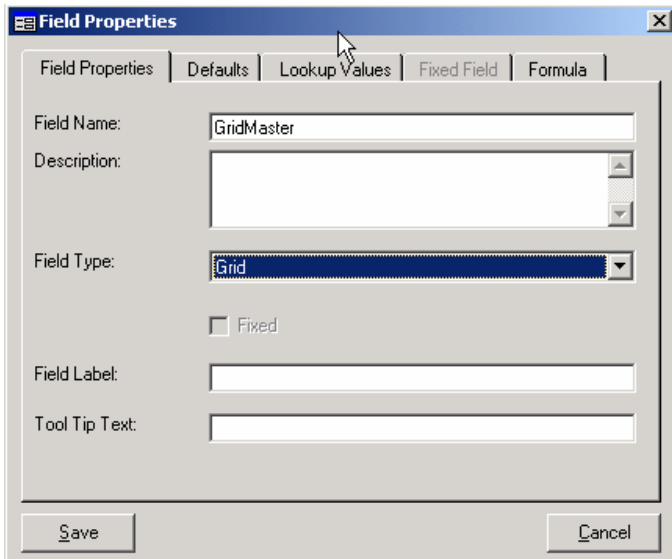
```
Public Sub Form_AfterLoad ()
    Call MakeWidgetEnabled("ContractConfigFields", "TSAuthoriser1",
"btnLookupFind", False)
End Sub

Private Sub MakeWidgetEnabled(sTableName, sFieldName, sAssociatedWidgetName,
bEnabledFlag)
    Set widget = FormObjectsLocal.ShowForm.GetFieldWidget((sTableName),
(sFieldName))
    If Not widget Is Nothing Then
        With
FormObjectsLocal.FormRef.Controls(sAssociatedWidgetName)(widget.index)
            .Enabled = False
            '.backcolor = vbRed
        End With
    End If
End Sub
```

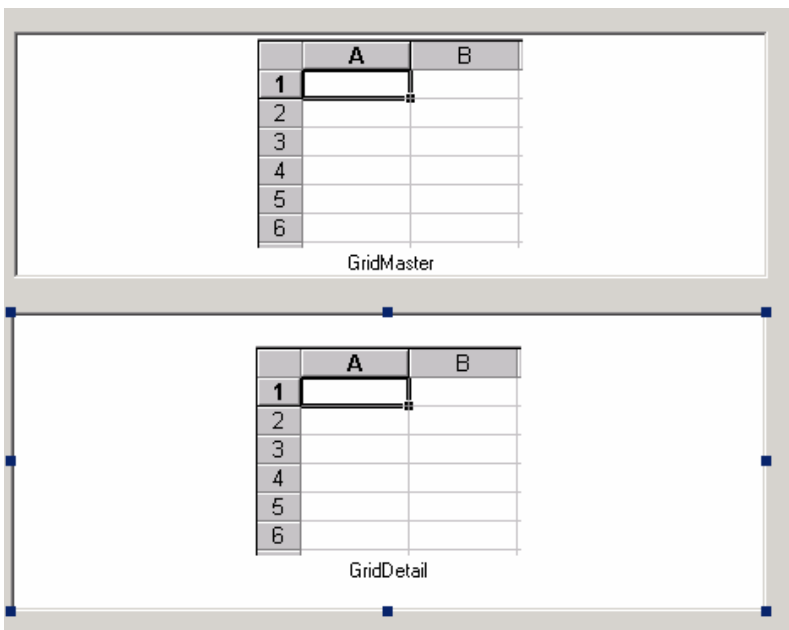
## 7. Step by Step Example of Master-Detail Dynamic Grid Handling

In this example, we create 2 grids using form designer on a form (or action) which are not linked to any entities and write script code to handle the interaction between them.

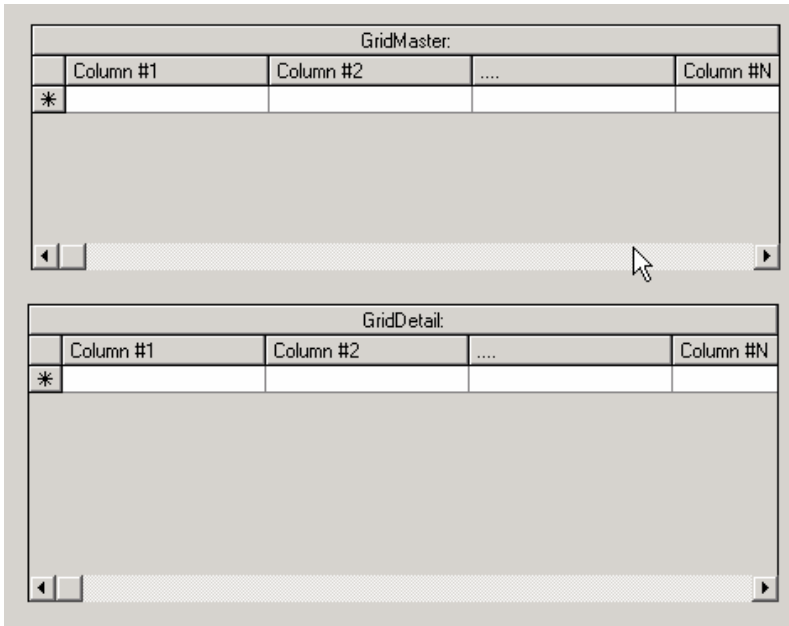
Step 1: Open form and create 2 grids in Form Designer.  
Call these GridMaster and GridDetail.



Step 2: Place these on a convenient location like so:



Step 3: Save form design and test the design. The tab should look as follows:



Step 4: Open the script editor for this form and paste the following code into the editor:

```
Public Sub Form_AfterLoad ()
    Call InitialiseGridMaster( _
        FormObjectsLocal.ShowForm.GetFieldWidget( _
            "RequirementConfigFields", "GridMaster"))
End Sub

' The table selection grid
Private Sub InitialiseGridMaster(dwGrid)
    Dim i

    If dwGrid Is Nothing Then
        MsgBox.appmsg "GridMaster Not found!"
        Exit Sub
    End If

    ' Initialise grid
    With dwGrid
        .Columns.RemoveAll
        .DataMode = 2                ' Add Item Grid
        .AllowAddNew = False
        .AllowUpdate = False
        .Caption = "Main TriSys Tables - Select to view details"
        .MaxSelectedRows = 1
        .RecordSelectors = True
        .SelectTypeRow = 1          'Single Select

        ' More settings
    End With

    ' Add columns
    With DataWidgets
        .AddColumnToGrid (dwGrid), 0, "Table Name", 4000, True
        .AddColumnToGrid (dwGrid), 1, "Actual Table Name", 2000, False
    End With

    ' Add Rows
```

```

With dwGrid
    .AddItem "@Contact@,@Contact@"
    .AddItem "@Company@,@Company@"
    .AddItem "@Placement@,@Contract@"
    .AddItem "@Requirement@,@Requirement@"
    .AddItem "@Users@,@Users@"

    .AddItem "@FieldDescription@,@FieldDescription@"
    .AddItem "@FormProperties@,@FormProperties@"
End With

' Select first row and fire detail event
dwGrid.AddItemBookmark 1
Call InitialiseGridDetail( _
    FormObjectsLocal.ShowForm.GetFieldWidget _
        ("CompanyConfigFields", "GridDetail"), "Contact")
End Sub

Public Sub Grid_RowSelection ()
    Set fd = FormObjectsLocal.ShowForm. _
        GetFieldDescriptionFromWidgetHwnd (FormObjectsLocal.EventControl.hWnd)
    If fd Is Nothing Then Exit Sub

    If fd.TableFieldName = "GridMaster" Then
        Set dwGrid = FormObjectsLocal.EventControl
        sTableName = dwGrid.Columns(1).Text
        Call InitialiseGridDetail(FormObjectsLocal.ShowForm.GetFieldWidget _
            ("CompanyConfigFields", "GridDetail"), sTableName)
    End If
End Sub

Private Sub InitialiseGridDetail(dwGrid, sTableName)
    Dim i, c

    If dwGrid Is Nothing Then
        MsgBox.appmsg "GridDetail Not found!"
        Exit Sub
    End If

    ' Initialise grid
With dwGrid
    .Columns.RemoveAll
    .DataMode = 2                ' Add Item Grid
    .AllowAddNew = False
    .AllowUpdate = False
    .Caption = "First 10 Rows of Table: " & sTableName
    .MaxSelectedRows = 1
    .RecordSelectors = True
    .SelectTypeRow = 1          'Single Select
    .Redraw = False
End With

    i = 0
    sSQL = "Select Top 10 * From " & sTableName
    Set rs = TriSysDatabase.OpenRecordset((sSQL))
    Do While Not rs.eof
        i = i+1
        If i = 1 Then
            ' Add columns
            For c = 1 to rs.Fields.Count
                DataWidgets.AddColumnToGrid (dwGrid), c-1, _
                    rs.Fields(c-1).Name, 1000, True
            Next
        End If
    End While
End Sub

```

```

' Add row to grid
sAddItemString = ""
For c = 1 to rs.Fields.Count
    If c > 1 Then sAddItemString = sAddItemString & ","
sAddItemString = sAddItemString & "@" & _
    StringFunctions.VBFormat(rs.Fields(c-1).Value, "@") & "@"
Next
dwGrid.AddItem sAddItemString

rs.MoveNext

Loop
rs.close

dwGrid.Redraw = True
End Sub

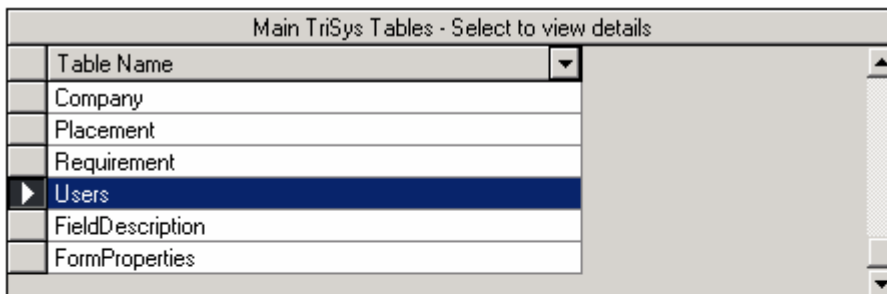
Public Sub Grid_DblClick ( Cancel )
Set fd = _
FormObjectsLocal.ShowForm.GetFieldDescriptionFromWidgetHwnd _
    (FormObjectsLocal.EventControl.hWnd)
If fd Is Nothing Then Exit Sub

If fd.TableFieldName = "GridDetail" Then
Set dwGrid = FormObjectsLocal.EventControl
sField1 = dwGrid.Columns(0).Text
MessageBox.appmsg "You double clicked record: " & sField1 & "?"
End If
End Sub

```

Step 5: Edit the script if necessary and replace CompanyConfigFields with the table upon which your created the grids.

Step 6: Apply the changes and open the TriSys form and it should operate as follows when any row in the grid is selected in the master grid (with either the mouse click or cursor key up/down), the detail grid should show the first 10 records from the table:



First 10 Rows of Table: Users							
UserId	LoginName	Password	Name	TelNo	LoggedIn	F	
105	GarryL	001	Garry	01223	False	C	
127	GrahamB		Graham		False	C	
129	JacquesP		Jacques		False	C	
125	JohnT		John		False	C	
126	LouiseB	001	Louise		False	C	
107	MattJ	001	Matt Jessop		False	C	
123	MauriceP		Maurice		False	C	
128	RoderB		Roder		False	C	

The code demonstrates a number of features:

- Creation of empty grids with no standard functionality
- Dynamic creation of grid columns
- Grid Event processing: row selection and row drill down
- Master/Detail functionality which can be applied to any database tables
- Speed at which the script engine can interact with the database in real-time
- Flexibility of independence from the TriSys entity builder framework

## 8. Field Visibility Example Code – How to Use FormFields within ShowForm

In order to fully address the issue of widget visibility from script, I attach some code which you will find useful.

Here is a technical explanation of the TriSys internal config forms engine:

Each CShowForm class (the run-time FormObjectsLocal.ShowForm object) has a number of child objects.

One of the objects is CFormField which is an on-form instance of the CFieldDescription class which describes the field. By setting the Visible (or Enabled, BackColour, FontSize, ForeColour, Left, Top, Height, Width, ReadOnly, ShowLabel, ToolTipText) property of the CFormField object, you allow CShowForm to treat the field in the way you want it which overrides the form designed attributes.

The example below shows how I have done it for the contact availability date which I do not want to be visible when the contact is a client.

The SetFieldVisible function can be placed on any form and used accordingly for any field.

```
' Hide/Show controls accordingly
Sub ProcessContactType()

    ' Determine whether the contact is a client or not
    bClient = Not FormObjectsLocal.ShowForm.GetFieldValue("Contact", _
        "ContactTypeCandidate")

    ' Now set the visibility of these fields
    Call SetFieldVisible("Contact", "AvailabilityDate", Not bClient)
End Sub

' Generic function to set a fields (and labels) visibility
Private Sub SetFieldVisible(sEntityName, sTableFieldName, bVisible)
    Dim objField, objLabel, formField

    With FormObjectsLocal.ShowForm
        ' Get the widget and CFormField object for the specified field
        Set objField = .GetFieldWidget((sEntityName), (sTableFieldName))
        Set formField = .GetFormFieldFromFieldName((sEntityName), (sTableFieldName))

        ' If field found, set visibility so that CShowForm class knows when to show
        If Not formField Is Nothing Then formField.Visible = bVisible

        ' Only set the visibility if this field is in view
        If FormObjectsLocal.ShowForm.isWidgetVisibleOnCurrentTab( _
            objField.hWnd) Then

            ' Make field widget visible/invisible right now
            ' without refreshing whole form
            If Not objField Is Nothing Then objField.Visible = bVisible

            ' Do control label visibility now too
            Set objLabel = .GetFieldWidget((sEntityName), (sTableFieldName), 1)
            If Not objLabel Is Nothing Then objLabel.Visible = bVisible
        end if
    End With
End Sub
```

----- Original Message -----

From: Arnold Lieberman <[arniel@justassociates.com](mailto:arniel@justassociates.com)>

To: GaRRy Lowther <[garry@technical.demon.co.uk](mailto:garry@technical.demon.co.uk)>

Cc: Graham Bridle <[grahamb@justassociates.com](mailto:grahamb@justassociates.com)>

Sent: 30 January 2001 17:33

Subject: Request for new facilities

> GaRRY,

> There are a couple of things that I need:

- > \* When a config form is clicked on, all the controls refresh themselves. If there is any script on the form to modify the controls' appearance, the changes are lost when the refresh is completed. Can I have a hook to intercept the refresh?
- > \* When selecting a placement to display from the requirement form, the placement form is loaded but the controls are not displayed until the form is clicked on. Can you investigate this? I can send you an export schema if that will help.

## 9. Button Visibility Example Code – How to Use FormButton within ShowForm

In order to fully address the issue of button visibility from script, here is some code which you will find useful. This is very similar to that in the previous chapter, but uses the form buttons collection.

```
Public Sub Button_Click
    ' Not interested in non-config buttons
    if FormObjectsLocal.EventControl.Name <> "btn" then Exit Sub

    Set btn = FormObjectsLocal.ShowForm.GetButtonFromIndex("btn", _
        FormObjectsLocal.EventControl.Index)
    If btn Is Nothing Then Exit Sub

    Select case btn.ButtonFunction
        Case "The One To Make Invisible"
            Call SetButtonVisibility(btn.ButtonFunction, False)
            Exit Sub
    End Select
End Sub

' Set the visibility of the specified button.
' This uses the CShowForm class to control the buttons visibility between
' tab clicks
'
Private Function SetButtonVisibility(sButtonFunction, bVisible)
    ' Get the form button from the CShowForm class
    Set formButton = FormObjectsLocal.ShowForm.GetButtonFromFunction( _
        "btn", (sButtonFunction))
    If formButton Is Nothing Then Exit Function

    ' Set its visibility
    formButton.Visible = bVisible

    ' Now set the current widget instance visibility
    Set btnWidget = formButton.FormWidget
    If btnWidget Is Nothing Then Exit Function

    ' Only set the visibility if this button is in view
    If FormObjectsLocal.ShowForm.isWidgetVisibleOnCurrentTab(btnWidget.hWnd) Then
        btnWidget.Visible = bVisible
    End If
End Function
```

## 10. Setting Selected Tab In Script

Developers,

To force another tab to get focus on a config form and force redraw of the tab widgets, use the following code

```
With FormObjectsLocal.FormRef
  .Tabby.Tabs(2).Selected = True
  .tabby_click
End With
```

This performs the same processing as if the user selected the tab with the mouse.

The .tabby\_click event corresponds to the Public Sub tabby\_Click method of the underlying form (you VB'ers).

Garry.

-----Original Message-----

**From:** Graham Bridle [<mailto:Grahamb@justassociates.com>]

**Sent:** 29 January 2001 09:52

**To:** '[garry@technical.demon.co.uk](mailto:garry@technical.demon.co.uk)'

**Subject:** Selecting Tabs

Garry,

I am using the following command to select different tabs on a form, it works, but it is not drawing the form objects until you click on the form somewhere.

Me ? or you ?

g

```
FormObjectsLocal.FormRef.Tabby.Tabs(2).Selected = True
```

## 11. Detecting Which Button is Pressed

Create a button using form designer and give it a function name: "TabTest"

Drop it onto a form.

Write the following script code to catch the button click even for it:

```
Public Sub Button_Click()
```

```
    ' Not interested in non-config buttons
```

```
    if FormObjectsLocal.EventControl.Name <> "btn" then Exit Sub
```

```
    Set btn = FormObjectsLocal.ShowForm.GetButtonFromIndex("btn", _  
        FormObjectsLocal.EventControl.Index)
```

```
    If btn is nothing then exit sub
```

```
    If btn.ButtonFunction = "TabTest" Then
```

```
        ' Your code for this button in here
```

```
    End if
```

```
End Sub
```

## 12. Firing an Action Button

This tiny script allows you to invoke an action from a config form.

Steps to reproduce:

1. Create action
2. Create an action button
3. Add the action button to the form
4. Write this code to fire the action associated with the button from script:

```
Sub YourFunction()  
    Call FireActionButton("Get In Touch")  
End Sub  
  
Private Sub FireActionButton(sActionName)  
  
    Set btn = FormObjectsLocal.ShowForm.GetButtonFromFunction("btn", (sActionName))  
    If btn Is Nothing Then Exit Sub  
  
    ' Fire the action button  
    btn.FormWidget.Value = True  
End Sub
```

See next example for more code relating to this.

### 13. Firing an Action From Script Code

This script allows you to invoke an action from any form using script code alone.

In the example, a button is created with no link to an action or extension. This is then fired to operate upon the details of an independent script-controlled grid to fire the action using the grid data.

This makes use of an exposed function on the underlying form:

```
FormObjectsLocal.FormRef.RunFormAction ActionId, ClientContactId, CandidateId,  
CompanyId, RequirementId,, PlacementI, TimeSheetId
```

Steps to reproduce:

1. Follow section above: **Step by Step Example of Master-Detail Dynamic Grid Handling** in order to create master/detail script controlled grids.
2. Create action (or reuse existing action e.g. Note/Call etc...) – Note the ActionId
3. Create a button called **RunActionAgainstContactId** – do **NOT** assign to an action.
4. Add the button to the form beneath the detail grid.
5. Write this code to fire the action associated with the button from script:

```
Public Sub Button_Click  
    ' Not interested in non-config buttons  
    if FormObjectsLocal.EventControl.Name <> "btn" then Exit Sub  
  
    Set btn = FormObjectsLocal.ShowForm.GetButtonFromIndex("btn", _  
        FormObjectsLocal.EventControl.Index)  
  
    If btn Is Nothing Then Exit Sub  
    If btn.ButtonFunction = "RunActionAgainstContactId" Then  
        Call RunActionAgainstContactId  
        Exit Sub  
    End If  
End Sub  
  
' Button clicked to fire an action against the currently selected contact  
,  
  
Private Sub RunActionAgainstContactId  
    Set dwGrid = FormObjectsLocal.ShowForm.GetFieldWidget(_  
        "CompanyConfigFields", "GridDetail")  
    If dwGrid Is Nothing Then Exit Sub  
  
    If instr(dwGrid.Caption, "Contact") = 0 Then  
        MsgBox.AppStopMsg _  
            "You can only fire this button when contacts are displayed in the grid."  
        Exit Sub  
    End If  
  
    ' Get selected contactid  
    ContactId = CLng(GetGridCellValue(dwGrid, "ContactId"))  
  
    'MessageBox.appmsg "Running action against Id: " & ContactId  
  
    ' ActionId=32 is Note
```

```
FormObjectsLocal.FormRef.RunFormAction 32, clng(ContactId), 0, 0, 0, 0, 0  
End Sub
```

## 14. Validating Action Values

```
' Example to demonstrate the capturing in script of the on-screen fields
' in an action so that the action can be prevented from running if all
' expected fields are not correctly input by the user.
'
' Feb 2001
'

' Validate input here
Public Sub Form_BeforeSaveRecord ( Id, Cancel )
    bCancel = Not ValidateActionFields

    If bCancel Then FunctionStatus.SetParameter("Cancel") = True
End Sub

Private Function ValidateActionFields()
    ' Check the candidate Id - must be set when creating a placement
    ' candidateId = FormObjectsLocal.ShowForm.GetFieldValue("ContractPlacement",
"CandidateId")
    Set candidateIdCollection = FormObjectsLocal.Formref.candidateIdCollection
    If candidateIdCollection Is Nothing Then
        bCandidateError = True
    Else
        If candidateIdCollection.count = 0 Then
            bCandidateError = True
        Else
            If clng(candidateIdCollection(1)) = 0 Then
                bCandidateError = True
            End If
        End If
    End If
    End If

    If bCandidateError Then
        MsgBox.AppStopMsg "Please select a candidate for the placement."
        Exit Function
    End If

    ' Make sure rates are set
    payRate = cdbl(FormObjectsLocal.ShowForm.GetFieldValue("Requirement",
"StandardPayRate"))
    If payRate <= 0 Then
        MsgBox.AppStopMsg "Please enter a valid candidate pay rate."
        Exit Function
    End If
    chargeRate = cdbl(FormObjectsLocal.ShowForm.GetFieldValue("Requirement",
"StandardChargeRate"))
    If chargeRate <= 0 Then
        MsgBox.AppStopMsg "Please enter a valid client charge rate."
        Exit Function
    End If

    ' Make sure we have valid dates
    startDate = FormObjectsLocal.ShowForm.GetFieldValue("Requirement",
"EarliestStartDate")
    If len(startDate) = 0 Then
        MsgBox.AppStopMsg "Please enter a start date for the placement."
        Exit Function
    End If
    endDate = FormObjectsLocal.ShowForm.GetFieldValue("Contract",
"ContractEndDate")
    If len(endDate) = 0 Then
        MsgBox.AppStopMsg "Please enter an end date for the placement."
```

```
        Exit Function
    End If

    ' All OK?
    ValidateActionFields = True
End Function
```

## 15. Processing Requirement Shortlist

When performing shortlisting using bespoke workflow processing (e.g. headhunting) the script may require access to the search results shortlist and delete check boxes which indicate which contacts the user has selected for shortlisting or deletion.

The script developer can change this column caption using script, but obtaining the actual selected rows is difficult because the grid does not perform as expected.

To get around this limitation, two extra functions have been exposed on the requirement form so that the script developer can get access to the respectively selected contacts in the same way that TriSys does internally.

The functions are:

GetShortlistedCandidatesFromSearchResults  
GetDeletedCandidatesFromSearchResults

Both return a collection of ContactId's of contacts where the user has selected the shortlist or delete columns.

The example below demonstrates the usage:

```
Public Sub Button_Click ()
    ' Not interested in non-config buttons
    if FormObjectsLocal.EventControl.Name <> "btn" then Exit Sub

    Set btn = FormObjectsLocal.ShowForm.GetButtonFromIndex("btn",
FormObjectsLocal.EventControl.Index)
    If btn Is Nothing Then Exit Sub
    If btn.ButtonFunction = "ScriptShowShortlisted" Then
        ' Get all folk shortlisted in grid
        Set candidateCollection = _
FormObjectsLocal.FormRef.GetShortlistedCandidatesFromSearchResults
        If candidateCollection Is Nothing Then Exit Sub

        ' Enumerate through collection to display to user
        ' (or whatever you like?)
        If candidateCollection.Count > 0 Then
            For i = 1 to candidateCollection.count
                If i > 1 Then s = s & ", "
                s = s & candidateCollection(i)
            Next
            MsgBox.appmsg "Selected CandidateId's: " & s & "?"

            ' Now capture the contacts into the 'special' table:
            'RequirementCandidateUnderScriptControl
            ' These will then cause the following refresh method to actually
            ' remove them from the search results

            For each v in candidateCollection
                ContactId = clng(v)
                MySQL = "Insert into " & _
                    "RequirementCandidateUnderScriptControl" & _
                    "(RequirementId, ContactId)" & _
                    " Values(" & m_RequirementId & ", " & ContactId & ")"
                TriSysDatabase.Execute (mysql)
            Next
        End If
    End If
End Sub
```

```
        ' Now refresh the grid to cause those selected to disappear
        btnIndex = _
FormObjectsLocal.ShowForm.GetButtonFromFunction("btn", "Search").Index
        FormObjectsLocal.FormRef.btn(btnIndex).Value = 1
    End If
End If
End Sub
```

## 16. Creating a Lookup Combo

This procedure takes the example of creating a job title combo on the company form and populating it with job titles. This is not possible to do using reference fields.

Steps to reproduce:

- a. Open form designer for company form and create a new config field called **JobTitles**.
- b. This should be of type Lookup, not searchable, and with no values.
- c. Drop this onto the company form
- d. Write the following piece of code into the script editor for the form:

```
Public Sub Form_AfterLoad
    ' JobTitles Combo
    Call InitialiseJobTitlesCombo _
        (FormObjectsLocal.ShowForm.GetFieldWidget( _
            "CompanyConfigFields", "JobTitles"))
End Sub

' JobTitles Combo
Private Sub InitialiseJobTitlesCombo(cmbJobTitles)

    If cmbJobTitles Is Nothing Then Exit Sub

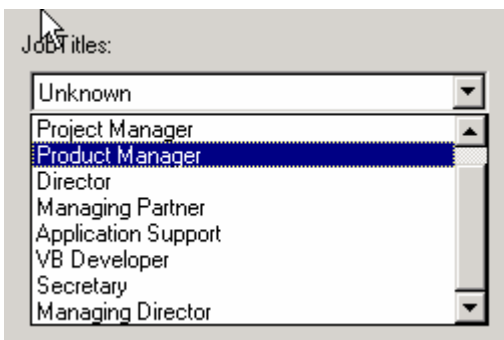
    cmbJobTitles.clear

    MySQL = "Select JobTitle From JobTitle" & _
        " Order by Ordering"
    Set rs = TriSysDatabase.OpenRecordset((MySQL))
    Do While Not rs.EOF
        sJobTitle = rs(0)

        With cmbJobTitles
            .AddItem sJobTitle
            If .ListCount = 1 Then .ListIndex = 0
        End With

        rs.MoveNext
    Loop
    rs.Close
End Sub
```

The widget on the form should look as follows when run in TriSys:



## 17. Asking the User to Choose from a List

This procedure shows how to use the internal frmSelection form to offer the user a choice of items in order to choose one.

```
Private Sub SelectionFormTest
    Set frmSelection = FormFunctions.GetFormReference("frmSelection")
    With frmSelection
        .ClearItems
        .AddItem "1. Do Nowt at " & Now
        .AddItem "2. Do Something at " & Now

        .Caption = "Selection Test"
        .label = "Please select any option"
        .ShowModal
        If Not .Selected Then Exit Sub
        MsgBox.AppMsg "You selected '" & .SelectedItem & "' ?"
    End With
End Sub
```

For a discussion and examples of using the more sophisticated SQL selection mechanism, see the following section.

## 18. Using SQL Selection Object

This procedure shows how to use the internal SQL selection object (5.00.063 or later) to pass in a SQL statement, setup grid columns etc and retrieve the selected records from a modal dialogue box. It is of use when the you need to offer the user a choice of selecting a range of records from any table or tables in the database.

```
' This function will allow the user to select a candidate contact from
' a pop-up and will show the correct priority and type to boot!
Private Sub SQLSelectionFormTest
    Set SQLSelection = Selection.GetSelectSQLObject
    With SQLSelection
        .Columns = 6
        .ColWidth(0) = 0
        .ColWidth(1) = 0
        .ColWidth(2) = 0
        .ColWidth(3) = 1500
        .ColWidth(4) = 2000
        .ColWidth(5) = 1200
        .Caption = "Select any Candidate"
        .SQL = "Select Top 50 ContactId, Priority, Type, Christian as Forenames, " & _
            "Surname, DateOfBirth as 'Date of Birth'" & _
            " From Contact" & _
            " Where Contact.Type = 1" & _
            " Order by Surname, Christian"

        .ContactSelection = True
        .PriorityColumn = 1
        .TypeColumn = 2
        If Not .Selection Then Exit Sub
        MsgBox.AppMsg "You selected ContactId:" & .SelectedRecord(1) & " ?"
    End With
End Sub

' SQL selection of any number of placements
Private Sub SQLSelectionPlacement
    Set SQLSelection = Selection.GetSelectSQLObject
    With SQLSelection
        .Columns = 4
        .ColWidth(0) = 0
        .ColWidth(1) = 1200
        .ColWidth(2) = 2000
        .ColWidth(3) = 2500
        .Caption = "Select one or more Placements"
        .SQL = "Select ContractId, Reference, Company.Name as Company" & _
            ", Christian + ' ' + Surname as Candidate" & _
            " From Contract, Company, Contact" & _
            " Where Contract.CompanyId = Company.CompanyId" & _
            " And Contract.CandidateId = Contact.ContactId" & _
            " Order by Reference"

        .MultipleSelection = True
        If Not .Selection Then Exit Sub

        For i = 1 to .SelectedCount
            If i > 1 Then s = s & ", "
            s = s & clng(.SelectedRecord(cint(i)))
        Next

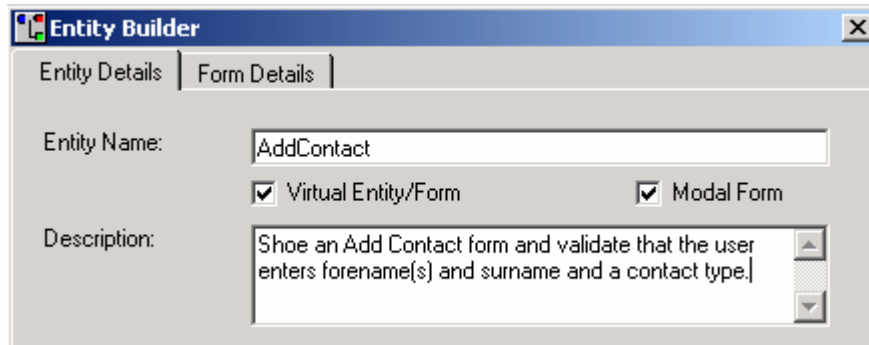
        If len(s) > 0 Then MsgBox.appMsg "You selected " & .SelectedCount & _
            " placements: " & s & "?"
    End With
End Sub
```

Note how you can set up the column widths, specify multiple record selection (using shift/ctrl keys), set caption, assign SQL statement, cater for contact priorities and contact types to allow colours.

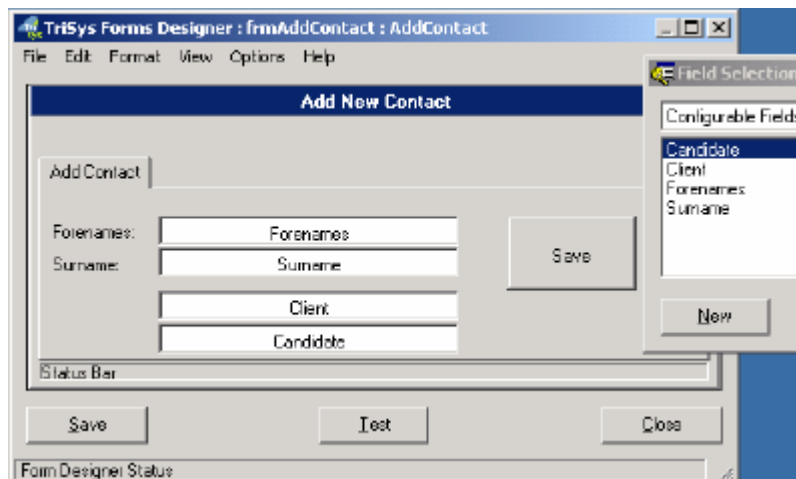
## 19. Modal Virtual Form Interaction

This procedure shows how to use the modal virtual form capability of TriSys (5.00.063 or later) by invoking it from script code to display defaults, validate the data and return the data to the caller.

Step 1: Create a modal virtual entity



Step 2: Design the modal virtual entity form and create data capture fields



Step 3: Design the modal virtual entity script code to capture data and validate it

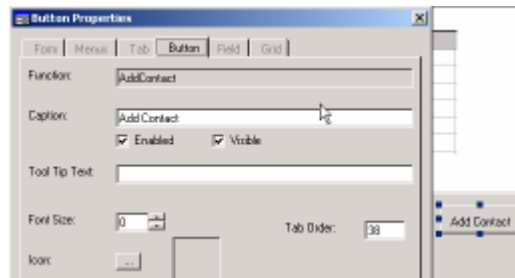
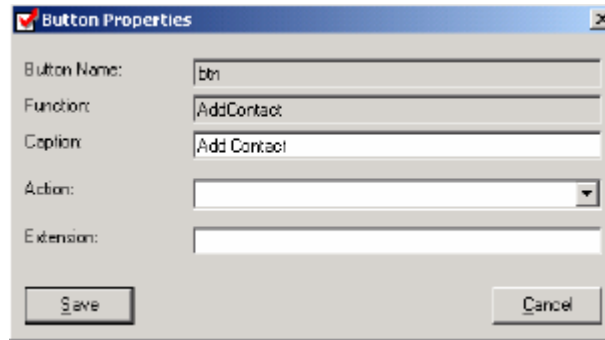


```
' On button click - validate input and close if OK
Public Sub Button_Click ()
    With FormObjectsLocal.showForm
        sForenames = .GetFieldValue("AddContactConfigFields", "Forenames")
        sSurname = .GetFieldValue("AddContactConfigFields", "Surname")

        If len(sForenames) = 0 or len(sSurname) = 0 Then
            MsgBox.AppStopMsg "Please enter both forenames and surname." & _
                sForenames & sSurname
        Exit Sub
    End If
End With

' All OK, then close form using correct exposed method to preserve stacked scripts
FormObjectsLocal.FormRef.FormUnload
End Sub
```

#### Step 4: Create a button on the calling form for the user



#### Step 5: Write code for the button to invoke the modal virtual form, pass in default values and capture validated values from it.

```
Public Sub Button_Click
    ' Not interested in non-config buttons
    if FormObjectsLocal.EventControl.Name <> "btn" then Exit Sub

    Set btn = FormObjectsLocal.ShowForm.GetButtonFromIndex("btn", _
        FormObjectsLocal.EventControl.Index)
    If btn Is Nothing Then Exit Sub
    If btn.ButtonFunction = "AddContact" Then
        Call AddContactButton
        Exit Sub
    End If
End Sub

' Show a modal virtual entity to capture contact details to add to our grid
Private Sub AddContactButton
    Set dwGrid = FormObjectsLocal.ShowForm.GetFieldWidget("CompanyConfigFields", "GridUpdate")

    If dwGrid Is Nothing Then Exit Sub

    Set frmModalForm = FormFunctions.GetFormReference("frmAddContact")
    If frmModalForm Is Nothing Then
        MsgBox.AppStopMsg "Could not find form: frmAddContact"
        Exit Sub
    End If

    With frmModalForm
        ' Send initialisation data in
        .SetFieldValue("Forenames") = StringFunctions.VBFormat(Now, "DDDD DD")
        .SetFieldValue("Surname") = StringFunctions.VBFormat(Now, "MMMM YYYY")

        ' Show it
        .ShowModal

        ' After modal form is closed, read the values from it
        sForenames = .GetFieldValue("Forenames")
        sSurname = .GetFieldValue("Surname")
    End With

    If len(sForenames) > 0 and len(sSurname) > 0 Then
        dwGrid.AddItem "@" & sForenames & "@,@" & sSurname & "@"
    End If
End Sub
```

Step 6: What it looks like in operation:

+

Add Contact

Forenames: Saturday 31

Surname: March 2001

Client

Candidate

+

Save

Step 7: This data is then added to the underlying form grid:

Updateable Grid	
Forename(s)	Surname
Saturday 31	March 2001
Saturday 31	March 2001
Saturday 31	March 2001
Sasassdasdturday	March 2001
Saturday 31	March 2001
Saturday 31	March 2001
Saturday 31	March 2001
11	March 2001
Saturday 31	March 2001

Show Data

Add Contact

## 20. Action Form Collection Manipulation

This procedure shows how to add a contact to the list of default contacts in an action collection. It also demonstrates how to use the FastContactSearch function to pop-up a contact selection dialogue.

```
Sub Form_AfterLoad
    Dim coll, i, s

    Set coll = FormObjectsLocal.formref.ContactIdCollection
    If coll Is Nothing Then Exit Sub
    If coll.count = 0 Then Exit Sub

    ' Add another client
    ContactId = Selection.FastContactSelection(True, False)
    If ContactId = 0 Then Exit Sub

    ' Add to collection
    FormObjectsLocal.formref.ContactIdCollection.Add cstr(ContactId)

    ' Show visuals
    FormObjectsLocal.ShowForm.SetFieldValue("CallClient", "ClientContactId") = _
        "< List: " & FormObjectsLocal.formref.ContactIdCollection.Count & _
        " Client Contact records >"
End Sub
```

## 21. Firing A Report From Script

Reports can be easily invoked from the print button or by a new button which is linked to a specific report. This works fine for standard entities or new entities, where all record processing is taken care of when set up in form designer by the forms engine.

If however, a virtual entity has been implemented, or the report requires specific pre-processing (as opposed to writing script inside a report form), then invoking the report from script is the appropriate method.

This section shows how this is done.

Step 1: Create an entity.

Step 2: Create a Crystal report which links the tables and configure this in forms designer (see reports design and tutorial document).

Step 3: Add a button onto the form, but do not set it up as a report button as it will defeat the object of this example!

Step 4: Copy this code into the form script editor. Note how we capture the loaded or saved record id and this is used to create a SQL IN clause which is passed into the PrintReport function of the Initialisation.Reports collection.

```
Dim m_AdvertisementId          ' as Long

Public Sub Form_AfterLoadRecord ( Id )
    m_AdvertisementId = id
End Sub

Public Sub Form_AfterSaveRecord ( Id )
    m_AdvertisementId = id
End Sub

Public Sub Button_Click
    ' Not interested in non-config buttons
    if FormObjectsLocal.EventControl.Name <> "btn" then Exit Sub

    Set btn = FormObjectsLocal.ShowForm.GetButtonFromIndex _
        ("btn", FormObjectsLocal.EventControl.Index)
    If btn Is Nothing Then Exit Sub
    If btn.ButtonFunction = "PrintReportFromScript" Then
        Call PrintReportFromScript
        Exit Sub
    End If
End Sub

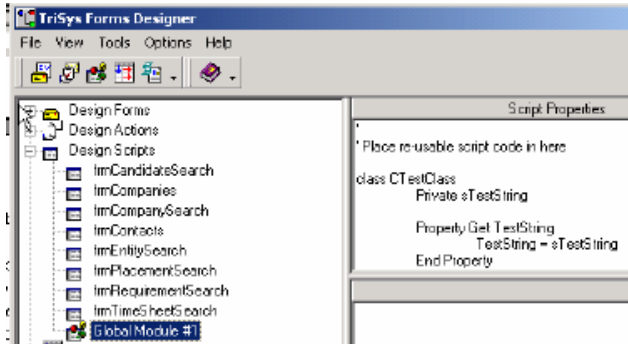
Private Sub PrintReportFromScript
    If m_AdvertisementId > 0 Then
        Initialisation.Reports.PrintReport "Advert Test.rpt", _
            "(Advertisement.AdvertisementId = " & m_AdvertisementId & ")"
    End If
End Sub
```

## 22. Using Classes in VBScript – Object Oriented Programming

VBScript version 5 is supported in TriSys. This adds sophisticated capabilities to VBScript, namely support for classes. Classes provide programmers with the opportunity to develop fully object oriented code by encapsulating re-usable business logic into a class. The class can be implemented to abstract the complexity of the logic by exposing a neatly designed interface which can be used by the programmer.

The potential of classes is shown in this very simple example:

Create a class inside a global module in form designer :



This is the code:

```
Class CTestClass
  Private sTestString

  Property Get TestString
    TestString = sTestString
  End Property

  Property Let TestString (s)
    sTestString = s
  End Property
End Class
```

The class is called CTestClass (classes should be prefixed by C). The class contains its own private variable calls sTestString which can only be accessed by users of the class using the properties defined. There are 2 properties, both called TestString which respectively Get and Set (via the Let operator) the value of the private variable. Private variables cannot be accessed directly by users of the class.

To see how the class might be used, see the following code:

```
Set m_TestClassObject = New CTestClass
m_TestClassObject.TestString = _
  StringFunctions.VBFormat(Now, "DD-MMM-YYYY HH:nn:ss")
MessageBox.appmsg "CTestClass.TestString=" & m_TestClassObject.TestString
```

The code creates a new instance of CTestClass thus creating an object (the definition of an object in OOP (object oriented programming) is that it is an instance of a class) which exposes all of the properties/functions/subroutines of the class.

Thus in the above example, we assign a value to the TestString property (Let), then display it (Get). Clearly the properties can do much more than simply set a value e.g. perform calculations, access the database, send messages etc...

When you access the TriSys objects e.g. [TriSysDatabase](#), this is actually an internal (VB6) object which is an instance of the class CDatabase.

## 23. Programming Grid Column/Cell Colours

The grid is a Sheridan Data Widget. Thus the properties and methods and events for the grid must follow this conventions demonstrated below.

This example processes the Grid\_RowLoaded event and colours a contact grid according to the normal contact colours for type and priority, but also colours rows of another grid where the cell is set to a particular value:

There are examples of how to do this in the ITContract database: Company and Contact forms. Look for Grid\_RowLoaded event.

This is how to set the row colours for any grid:

```
Sub Grid_RowLoaded
  With FormObjectsLocal.EventControl
    .Columns(2).BackColor = vbRed
    .Columns(2).ForeColor = vbBlue
  End With
End Sub
```

And for a tighter control, you can do this:

```
Public Sub Grid_RowLoaded ()
  Set dwGrid = FormObjectsLocal.EventControl
  If dwGrid Is Nothing Then Exit Sub
  Set fd = _
  FormObjectsLocal.ShowForm.GetFieldDescriptionFromWidgetHwnd (dwGrid.hWnd)
  If fd Is Nothing Then Exit Sub

  Select Case fd.TableFieldName
    Case "GridDetail"
      If Instr(dwGrid.Caption, "Contact") > 0 Then
        Call DataWidgets.ColourContactPriorities ((dwGrid), _
          Cint(GetGridCellValue(dwGrid, "Type")), _
          Cint(GetGridCellValue(dwGrid, "Priority")))
      End If

    Case "GridUpdate"
      ' Colour the row with my name in it to blue with yellow text
      sName = GetGridCellValue(dwGrid, "Surname")
      If Instr(sName, "Lowther") > 0 Then
        styleSetName = "Style" & vbYellow & vbBlue

        With dwGrid
          .StyleSets((styleSetName)).ForeColor = vbYellow
          .StyleSets((styleSetName)).BackColor = vbBlue

          For i = 0 To .Columns.Count - 1
            .Columns(i).CellStyleSet ((styleSetName))
          Next
        End With
      End If
    End Select
End Sub

Private Function GetGridCellValue(dwGrid, ColumnName)
  With dwGrid
    For i = 0 to .Columns.Count - 1
      If lcase(.columns(i).Caption) = lcase(ColumnName) Then
        GetGridCellValue = .Columns(i).Value
        Exit Function
      End If
    Next
  End With
End Function
```

```
End With  
End Function
```

This last example uses the knowledge about contact types to colour the grid based upon the users preferences.

## 24. Multi-Select Grid Rows

In order to create a grid with multi-select rows, the `.SelectTypeRow` property must be set to either 2 or 3:

```
' Initialise grid columns
With dwGrid
    .Columns.RemoveAll
    .DataMode = 2                ' Add Item Grid
    .AllowAddNew = False
    .AllowUpdate = False
    .Caption = "Placements for " & sSalesOffice
    .MaxSelectedRows = 8
    .RecordSelectors = True
    .SelectTypeRow = 2          ' 1 - single, 2 - multi
End With
```

In order to select all highlighted rows into an array, use this code:

```
m_NumSelectedPlacementIds = 0

With dwGrid
    ' Multi select - Get in order as appear in grid
    For r = 0 To .Rows - 1
        bm = .AddItemBookmark(r)
        For i = 0 To dwGrid.SelBookmarks.Count - 1
            If CStr(bm) = CStr(.SelBookmarks(i)) Then
                m_NumSelectedPlacementIds = m_NumSelectedPlacementIds + 1
                ReDim Preserve m_SelectedPlacementIds _
                    (Cint(m_NumSelectedPlacementIds))
                m_SelectedPlacementIds(m_NumSelectedPlacementIds) = _
                    CLng(.Columns(0).cellValue(bm))
            End If
        Next
    Next
End With
```

## 25. Launching Core Entity Forms From Script

If a grid is under script control and the user is permitted to drill down into the underlying record, these functions should be used :

```
If FormFunctions.isformloaded("frmPerson") Then
    Set f = FormFunctions.GetAFormInstance("frmPerson", True)

    ' Make it have focus
    Set frmMainBinder = FormFunctions.GetAFormInstance("frmMainBinder", True)
    Call frmMainBinder.AddFormToTabView((f), "Contact")
Else
    ' Load new form then
    Set f = FormFunctions.LoadFormByName("frmPerson", clng(ContactId), _
        "Script Contact")
End If
f.LoadContactDetails clng(ContactId)

' Load entity form using same technique
If FormFunctions.isformloaded("frmContractorInvoice") Then
    Set f = FormFunctions.GetAFormInstance("frmContractorInvoice")
Else
    ' Load new form then
    Set f = FormFunctions.LoadFormByName ("frmContractorInvoice", 1, _
        "Script ContractorInvoice")
End If
f.LoadEntityDetails 1
```

Or for entity forms

```
If FormFunctions.isformloaded("frmAdvertisement") Then
    Set f = FormFunctions.GetAFormInstance("frmAdvertisement")
    f.LoadEntityDetails 0
    FormFunctions.AddFormToTabView (f), ""
Else
    Set f = FormFunctions.LoadFormByName("frmAdvertisement", 0, "New Advert")
End If
```

This is an example of how to use a global function in a global module to encapsulate this complexity:

```
*****
' This Routine is called when any form in TriSys needs to be opened.
' Edited by Garry Lowther on 20 April 2002
*****
Public Sub gOpenForm(strFormName, intID)
    Dim f, bLoadRecord

    bLoadRecord = True          ' In most cases, we explicitly load the record
    If FormFunctions.isformloaded("frm" & strFormName) Then
        Set f = FormFunctions.GetAFormInstance("frm" & strFormName, True)

        ' Form exists so focus upon the form and associated tab
        Set frmMainBinder = FormFunctions.GetAFormInstance("frmMainBinder", True)
        Call frmMainBinder.AddFormToTabView((f), "Script" & strFormName)
    Else
        ' Load new form then
        Set f = FormFunctions.loadformbyname( _
            "frm" & strFormName, clng(intID), "Script" & strFormName)
        If clng(intID) > 0 Then bLoadRecord = False
        ' Will have loaded the record already
    End If

    ' Now load the record
    If Not f Is Nothing and bLoadRecord Then
        Select Case strFormName
            Case "Company":      f.LoadCompanyDetails CLng(intID)
            Case "Person":      f.LoadContactDetails CLng(intID)
            Case "Requirement":  f.LoadRequirementDetails CLng(intID)
            Case "Contract":     f.LoadPlacementDetails CLng(intID)
            Case "Advert":      f.LoadAdvertDetails CLng(intID)
            Case Else:          MsgBox.AppMsg "Unknown form: " _
                & strFormName
        End Select
    End If
End Sub
```

## 26. Using FileSystemObject in VBScript

The file system object component (*SCRRUN.DLL*) needs to be installed and registered for it to be used in script.

This gives access to the FileSystemObject which exposes many file system features. Search MSDN to find complete list. (*"Microsoft Scripting Run-time Library Features"*)

This example demonstrates how to use the object from a TriSys VBScript function to get attributes from a folder and file and to copy a single file:

```
Private Sub CopyFileUsingFileSystemObject
    Dim fso, fldr, f

    Set fso = CreateObject("Scripting.FileSystemObject")
    If fso Is Nothing Then
        MsgBox.appstopmsg "Could not instantiate the Scripting.FileSystemObject"
        Exit Sub
    End If

    Set fldr = fso.GetFolder("c:")
    MsgBox.AppMsg "Folder name: " & fldr.Name

    Set f = fso.GetFile("c:\Winnt\TriSys32.ini")
    MsgBox.AppMsg "Last modified: " & f.DateLastModified

    ' Copy file
    fso.CopyFile "c:\winnt\TriSys32.ini", "c:\temp\Copied.ini"
End Sub
```

## 27. Read all Grid Rows

If a grid is scripted, i.e. one that you have total control over?

If so it is known as an AddItem grid. Other grid population mechanisms are known as Bound (bound to a SQL recordset) and Unbound (grid calls you back for each row of data it requires). Standard TriSys grids usually use Unbound as they are more efficient.

Scripting currently cannot populate grids using any technique other than AddItem.

This example demonstrates reading a single column from all rows in the grid:

```
Dim strAdvertIdList, bm      ' Bookmark

strAdvertIdList = ""
With dwGrid
    For row = 1 To .Rows
        bm = .AddItemBookmark(row - 1)
        if len(strAdvertIdList) > 0 then strAdvertIdList = strAdvertIdList & ", "
        strAdvertIdList = strAdvertIdList & .Columns(0).CellValue(bm)
    Next
End With
```

## 28. External Document Launching

These functions are useful for launching the operating system features to open browsers, e-mail, auto-dial and documents:

```
' Fire E-Mail Client to send mail to recipient
Miscellaneous.EmailHotLink(sEmailAddress as String)

' Fire Browser to view address
Miscellaneous.WorldWideWebHotLink(sWWWAddress as String)

' Play specified .wav file
Miscellaneous.PlaySound(sWaveFile as String)

' Dial specified number using TAPI
Miscellaneous.AutoDialNumber(sTelNumber as String)

' Fire Shell Association program
ShellExecute.OpenThisDoc(hWnd as long, sFilename as string)
```

In this last function, you will need to pass in the actual window handle.  
Do this as follows:

```
' Check if file exists
if FileSystem.FileExists(sFilePathAndName) then
    ' Open the specified file.
    ' This function will open word for .DOC's, IE for .jpg, gif etc...,
    ' media player for .mpg etc...
    ShellExecute.OpenThisDoc FormObjectsLocal.FormRef.hWnd, (sFilePathAndName)
end if
```

## 29. Global Parameter Value

Each script runs in its own address space for the current form.

The only ways to share variables between forms/scripts is to either communicate through the database (slow/inefficient) or via formobjectsglobal and access form widgets (cumbersome).

A neater solution is to have a shared collection of parameters which can be accessed from any script direct into/out of memory.

This is accomplished by a let/get property on the Initialisation object:

```
Public Property Let GlobalParameterValue(sParameterName as String, sValue as Variant)
Public Property Get GlobalParameterValue(sParameterName as String) as Variant
```

This is used in the following example:

```
' Assign a variable to the global parameters
Initialisation.GlobalParameterValue("PlacementId") = 100

.....

' Retrieve a variable from the global parameters from any form script
PlacementId = cLng(Initialisation.GlobalParameterValue("PlacementId"))
```

### 30. Launching E-Mail Client

In order to send an e-mail from script, an object reference to the internal EMailGateway object must be used. This provides a generic encapsulation of the various e-mail clients and an easy to use set of properties and methods which allow for e-mails to be sent to many recipients with attachments.

This example demonstrates the mechanism:

```
Sub SendEMail
    Const RecipientTo = 1
    Const RecipientCC = 2
    Const RecipientBCC = 3
    Const ProtocolMAPI = 1
    Const ProtocolOutlook = 4
    Const ProtocolMailTo = 5
    Const ProtocolNotes = 6

    ' Get the e-mail address and attachment to send
    sEmailAddress = FormObjectsLocal.ShowForm.GetFieldValue( _
        "ReferenceConfigfields", "RefereeEmail")

    sAttachment = FormObjectsLocal.ShowForm.GetFieldValue( _
        "ReferenceConfigfields", "ReferenceRequest")

    sCandidateName = FormObjectsLocal.ShowForm.GetFieldValue( _
        "ReferenceRequest" , "CandidateId")

    ' Get the normal TriSys Email object references from TriSysEMailGateway.dll
    Set eMailObj = EMailGateway

    If eMailObj Is Nothing Then Exit Sub
    With eMailObj
        .EMailProtocol = ProtocolMAPI
        .NewMessage
        .AddRecipient sEmailAddress, sCandidateName, RecipientTo

        .Subject = "Reference request for " & sCandidateName

        .Message = "Please find attached a request for a reference."
        .AddAttachment sAttachment

        .ShowDialogue = True
        .Send (sErrStr)
    End With
End Sub
```

The following example is a real-world example of how to create a set of e-mails direct to candidates from the candidate search giving them the log in name and password to connect to the web site. The script should be called from a new button added to the action form.

```
' Manually invoke e-mail to send web password to a list of candidates.
' Jan 2002
' Garry Lowther
'

Public Sub Button_Click ()
    ' Not interested in non-config buttons
    If FormObjectsLocal.EventControl.Name <> "btn" Then Exit Sub
```

```

        Set btn = FormObjectsLocal.ShowForm.GetButtonFromIndex("btn",
FormObjectsLocal.EventControl.Index)
        If btn Is Nothing Then Exit Sub
        If btn.ButtonFunction = "MergeFromScript" Then Call SendEMail
End Sub

Sub SendEMail
    Const RecipientTo = 1
    Const RecipientCC = 2
    Const RecipientBCC = 3
    Const ProtocolMAPI = 1
    Const ProtocolOutlook = 4
    Const ProtocolMailTo = 5
    Const ProtocolNotes = 6

    Set candidateIdCollection = FormObjectsLocal.Formref.candidateIdCollection
    If candidateIdCollection Is Nothing Then Exit Sub
    If candidateIdCollection.Count = 0 Then Exit Sub

    ' Get the normal TriSys Email object reference
    Set eMailObj = EMailGateway
    If eMailObj Is Nothing Then Exit Sub
    With eMailObj
        .EmailProtocol = ProtocolOutlook

        ' Loop for each candidate in the list which has a valid e-mail address
        For each v in candidateIdCollection
            candidateId = clng(v)
            EMailAddress="": Forenames="": Surname="": WebPassword=""
            Call GetContactEMailDetails(candidateId, EMailAddress, Forenames,
Surname, WebPassword)
            If candidateId > 0 Then
                CandidateName = Forenames & " " & Surname

                .NewMessage
                .AddRecipient (EMailAddress), (CandidateName), RecipientTo

                .Subject = Initialisation.User.Name & " E-Mail to " &
CandidateName

                .Message = "Dear " & Forenames & vbcrLf & vbcrLf & _
                    "Login Name: " & CandidateName & vbcrLf & _
                    "Password: " & WebPassword & vbcrLf & vbcrLf & vbcrLf
& _
                    "Regards" & vbcrLf & _
                    Initialisation.User.Name

                .ShowDialogue = True
                .Send (sErrStr)

                ' TODO: Note/History + Follow ups if necessary?
            End If
        Next
    End With
End Sub

Private Sub GetContactEMailDetails(ContactId, EMailAddress, Forenames, Surname,
WebPassword)
    If ContactId = 0 Then Exit Sub

    MySQL = "Select EMail, Christian, Surname, WebPassword" & _
        " From Contact Left Join ContactConfigFields" & _
        " On Contact.ContactId = ContactConfigFields.EntityId" & _

```

```
        " Where Contact.ContactId = " & ContactId
Set rs = TriSysDatabase.openrecordset((MySQL))
If Not rs.eof Then
    On Error Resume Next      ' Ignore NULLS
    EMailAddress = rs.Fields("EMail")
    Forenames = rs.Fields("Christian")
    Surname = rs.Fields("Surname")
    WebPassword = rs.Fields("WebPassword")

    End If
    rs.close
End Sub
```

### 31. Colouring Tabs

It may be necessary to colour the tabs on data entry forms for data validation or to draw attention to certain tabs or even to satisfy the artistic design tendencies of customers!

This is how to do this:

```
set tab = FormObjectsLocal.FormRef.tabby.Tab(1)           ' First tab

with tab
  ' Set the tab back colour
  .BackColor = vbRed           ' Or any RGB colour
  .BackColorSource = 1         ' Important - it can't work this out on its own!

  ' Set the tab foreground colour - the text
  .ForeColor = vbYellow       ' Or any RGB colour
  .ForeColorSource = 1        ' Important - it can't work this out on its own!
end with
```

## 32. Crystal Report Form Script

When a report has been designed to have an associated data input form to capture parameters (See Reports Design Document and Tutorial), it is possible (build 5.00.081 or later) to manipulate the complete query before this is passed to the report for execution.

This allows the script developer to manipulate the query and add/remove portions of the SQL query depending upon user input. This example is taken from the report tutorial:

This example demonstrates the mechanism:

```
' This function is called after the user has requested the print of the report having filled in
' all on-screen values.
'
' This is the query from TriSys when invoked:
'
'Select
'*
'From
' ((Requirement Inner Join Contact Contact On
'Requirement.ContactId = Contact.ContactId)
'Inner Join Users On
'Requirement.UserId = Users.UserId)
'Inner Join Company On
'Requirement.CompanyId = Company.CompanyId
'
'Where 1=1
' And Requirement.RequirementId In
' (12,13,7,16,90,42,19,48,60,61,64,65,66,70,71,72,88,92,91,6,32,34,14,21)
' Or Requirement.RequirementId In ' (77,10,4,94,96,95,1,39,40,44,68,79,82,83,85,86,87,5,93,11)
'Order By Requirement.Reference ASC

Public Sub Form_AfterSaveRecord ( Id )
    'MessageBox.appmsg "Pre Tweak: " & vbCrLf & ReportProperties.SQLQuery

    ' Tweak the formula now
    ReportProperties.SQLQuery = RemoveRequirementIdInClause

    ' Get the input reference
    sReference = FormObjectsLocal.ShowForm.GetFieldValue("RequirementsListConfigFields", "AField")
    If len(sReference) > 0 Then
        ReportProperties.SQLQuery = _
            AddAndClause("And Requirement.Reference Like '%" & sReference & "%'")
    End If

    'MessageBox.appmsg "Post Tweak: " & vbCrLf & ReportProperties.SQLQuery
End Sub

Private Function RemoveRequirementIdInClause()
    Dim x, y
    Const AndRequirementIdIn = "And Requirement.RequirementId In ("
    Const OrRequirementIdIn = "Or Requirement.RequirementId In ("

    sFormula = ReportProperties.SQLQuery

    Do While True
        x = instr(sFormula, AndRequirementIdIn)
        If x = 0 Then x = instr(sFormula, OrRequirementIdIn)
        If x = 0 Then Exit Do

        ' Now find matching closing bracket
        y = instr(x, sFormula, ")")

        ' Get pre and post string segments
        sPre = Left(sFormula, x - 1)
        sPost = Mid(sFormula, y+1)

        ' Assemble again
        sFormula = sPre & sPost
    Loop

    RemoveRequirementIdInClause = sFormula
End Function
```

```
Private Function AddAndClause(sAndClause)
    Dim sFormula

    sFormula = ReportProperties.SQLQuery

    ' Find Order by
    x = instr(sFormula, "Order By")
    If x > 0 Then
        sFormula = Mid(sFormula, 1, x-1) & " " & sAndClause & " " & Mid(sFormula, x)
    End If

    AddAndClause = sFormula
End Function
```

### 33. Automating Word

Script can access the same word processor classes as used by TriSys. This is exposed through the WordProcessor object as the following example demonstrates:

```
Private Sub WordMagic
    With WordProcessor
        .Initialise False
        .OpenFile "D:\CVData\Upload\File1.doc", False, True
        .AppendFile "D:\CVData\Upload\File2.doc", True
        .SaveAs "D:\CVData\Upload\" & StringFunctions.VBFormat(Now, _
            "YYYYMMDDhhnnss") & ".rtf"
    End With
End Sub
```

In the above example, File1 is opened in word and File2 is appended to it. This is then saved as an RTF file with the current date/time stamp.

Other useful functions for automating word are:

Public Sub CloseFile(sFileName as String)	‘ Close named file
Public Property Get Description() As String	‘ Get name of word processor version
Public Property Get GetFileContents() As String	‘ Get unformatted text from open document
Public Property Let SetFileContents(s As String)	‘ Set unformatted text in open document
Public Sub HighlightWord(sWord as String)	‘ Highlight word in open document
Public Sub MailMerge(sDataSource as String)	‘ Mail merge current document using data source
Public Sub NewFile()	‘ Open a new document
Private Sub ReplaceInSection(section As Object, sOldWord As String, sNewWord As String)	‘ Replace old word with new word in named section

### 34. Searching for Requirements within Post Code Radius

This function has been exposed explicitly for script. It performs a search of the QAS post code database and returns a collection of requirements within a specified mile radius of a post code.

The example shows a function called which displays the list of requirement Id's:

```
Private Sub ViewAllRequirements
    ' Get a post code
    With BigInput
        .Text = "E1 1LU"
        .Caption = "Enter a valid post code?"
        .ShowModal
        If .Cancel Then Exit Sub
        sPostCode = Trim(.Text)
    End With

    With BigInput
        .Text = "10"
        .Caption = "Enter miles radius of " & sPostCode & "?"
        .ShowModal
        If .Cancel Then Exit Sub
        lRadius = Trim(.Text)
        If Not IsNumeric(lRadius) Then Exit Sub
    End With

    ' Call it now
    Set requirementIdCollection = _
        Initialisation.Requirements.GetRequirementCollectionWithinPostCodeRadius _
            ((sPostCode), clng(lRadius))

    If requirementIdCollection Is Nothing Then Exit Sub

    ' Display it
    sMessage = "Found " & requirementIdCollection.Count & " Requirements:" & vbCrLf
    For i = 1 to requirementIdCollection.Count
        sMessage = sMessage & " " & requirementIdCollection(i)
    Next

    MsgBox.AppMsg (sMessage)
End Sub
```

### 35. Updateable Grid for Data Entry

This section demonstrates how to create an updateable grid and use it for data input purposes:

Create a new field of type Grid called: GridUpdate.



Create a button to retrieve the grid data with button function: GridUpdateButton.

Open script editor to write code to initialise the grid:

```
' Editable grid
Call InitialiseGridUpdate( _
    FormObjectsLocal.ShowForm.GetFieldWidget("CompanyConfigFields", _
        "GridUpdate"))
```

....

```
' The editable grid which can capture data input and validate it.
' Can also allow add new rows if necessary.
```

```
Private Sub InitialiseGridUpdate(dwGrid)
    Dim i

    If dwGrid Is Nothing Then
        MsgBox.appmsg "GridUpdate Not found!"
        Exit Sub
    End If

    ' Initialise grid
    With dwGrid
        .Columns.RemoveAll
        .DataMode = 2                ' Add Item Grid
        .AllowAddNew = True
        .AllowUpdate = True
        .Caption = "Updateable Grid"
        .MaxSelectedRows = 1
        .RecordSelectors = True
        .SelectTypeRow = 1          'Single Select
    End With

    ' Add columns
    With DataWidgets
        .AddColumnToGrid (dwGrid), 0, "Forename(s)", 1500, True
        .AddColumnToGrid (dwGrid), 1, "Surname", 1500, True
    End With

    ' Add Rows
    With dwGrid
        .AddItem "@Garry@,@Lowther@"
        .AddItem "@Graham@,@Bridle@"
    End With
End Sub

' Capture button click
```

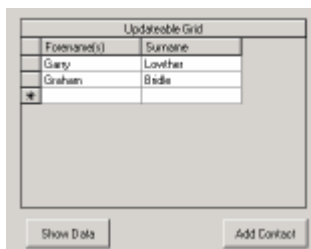
```
If btn.ButtonFunction = "GridUpdateButton" Then Call GridUpdateButton
```

...

```
' Called when the button is pressed to see the data
```

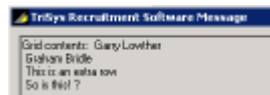
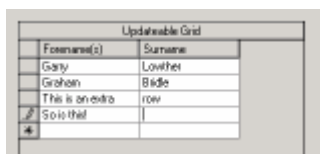
```
Private Sub GridUpdateButton  
    Set dwGrid = FormObjectsLocal.ShowForm.GetFieldWidget( _  
        "CompanyConfigFields", "GridUpdate")  
    If dwGrid Is Nothing Then Exit Sub  
  
    On Error Resume Next  
    With dwGrid  
        ' Important to force save of existing updated data  
        .Update  
  
        For row = 1 To .Rows  
            bm = .AddItemBookmark(row - 1)  
  
            If row > 1 Then s = s & VbCrLf  
            For col = 0 To .Columns.Count - 1  
                s = s & " " & .Columns(col).cellValue(bm)  
            Next  
        Next  
    End With  
  
    MsgBox.appmsg "Grid contents: " & s & "?"  
End Sub
```

Now run TriSys and open the form, the initial data should appear as follows:



You can then type into any row and update the text.

To add new rows, simply place the cursor into any cell in the row marked with an asterisk and all further rows can be added this way as shown below:



Pressing the Show Data button will show the input data as follows:

In order to capture when user is inputting data, write code in the Grid\_AfterUpdate event :

```
Grid | AfterUpdate  
Public Sub Grid_AfterUpdate | |  
    If FormObjectsLocal.eventcontrol Is FormObjectsLocal.ShowForm.GetFieldWidget("CompanyConfigFields", "GridUpdate")  
        MsgBox.appmsg "You updated the grid?"  
    End If  
End Sub
```

### 36. Programmed Fields Re-Appearing on Record Save

The problem with saving from tabs and causing fields under script control to show incorrectly can be a problem in certain circumstances.

The simple fix is to write code for the form after save record event which will force the first tab into focus, or if this is not acceptable, then the following script sample demonstrates how to cause the tabs to redraw the visible fields so that the fields do not show up:

```
' Called after record has been saved.
Public Sub Form_AfterSaveRecord ( Id )

' Force a refresh of the currently selected tab so that programmed fields do not
' show up on save when curren tab is not the first.
With FormObjectsLocal.FormRef
    selectedIndex = .tabby.selectedtab.index
    If selectedIndex <> 1 Then
        .Tabby.Tabs(1).Selected = True
        .Tabby.Tabs(selectedIndex).Selected = True
        .tabby_click
    End If
End With
End Sub
```

### 37. AddItem Grid Rows

Build 5.00.089 has easy to use functions to add fields and entire recordsets to a grid:

```
DataWidgets.AddItem (dwGrid As SSDBGrid, ParamArray FieldList())
DataWidgets.AddRecordsetRow (dwGrid As SSDBGrid, rs As Recordset)
DataWidgets.PopulateGridFromSQLQuery (dwGrid As SSDBGrid, sSQL as String)
DataWidgets.InitialiseAddItemGrid (dwGrid As SSDBGrid, sCaption as string)
```

This function is passed the handle to a grid and a list of values for each column:

```
` Initialise grid columns, SQL etc..
Set rs = TriSysDatabase.OpenRecordset( (sSQL) )
Do while not rs.EOF
    DataWidgets.AddItem (dwGrid), rs("ContactId"), rs("FullName")
    rs.MoveNext
Loop
rs.Close
```

If the recordset contains all possible columns and you do not want to do any field manipulation (formatting, lookups), another function can be used which accepts a recordset:

```
` Initialise grid columns, SQL etc..
Set rs = TriSysDatabase.OpenRecordset( (sSQL) )
Do while not rs.EOF
    DataWidgets.AddRecordsetRow (dwGrid), (rs)
    rs.MoveNext
Loop
rs.Close
```

Both of these functions remove the need to manipulate strings and field delimiters in order to add a row to a grid.

The ultimate function is `PopulateGridFromSQLQuery` which requires only a reference to the grid and a SQL statement in order to fully populate itself. It even generates the columns in the grid from the query if none exist.

```
` Get the top 50 companies ordered by company name
sSQL = "Select Top 50 * From Company Order by Name"
DataWidgets.PopulateGridFromSQLQuery (dwGrid), (sSQL)
```

This one function call populated the entire grid from a SQL statement.

A full (very small) example of how to populate a grid created in Form Designer is:

```
Public Sub Form_AfterLoad()

    Set dwGrid = FormObjectsLocal.ShowForm.GetFieldWidget( _
        "EntityName", "GridFieldName")
    if dwGrid is nothing then exit sub

    With DataWidgets
        .InitialiseAddItemGrid (dwGrid), "Top 50 Contacts"
        .PopulateGridFromSQLQuery (dwGrid), "Select top 50 * From Contact"
    End With
End Sub
```

## 38. Unbound Virtual Grid Population

Build 5.00.090 has easy to use functions to bind a SQL statement to a grid to provide:

- Virtual unbound recordsets – the data is demand paged from the server
- Sortable columns – standard column sorting (right mouse column header click)
- Automatic column generation – for prototyping purposes, all columns are generated

The techniques demonstrated here should be used for all grids which can be fully populated from a SQL statement.

The following new functions are now available:

```
DataWidgets.InitialiseUnboundGrid (dwGrid As SSDBGrid, sCaption as string)
```

```
DataWidgets.PopulateGridFromSQLQuery (dwGrid As SSDBGrid, sSQL as String)
```

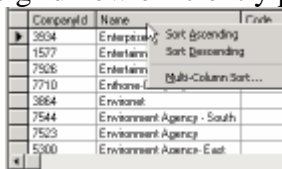
**Note:** the word *unbound* derives from the 3 modes that the data grid can operate:

- Add Item – rows are added to the grid 1 at a time like a list box. This is used for *small* non-SQL oriented data sets.
- Bound – the grid is bound to a data control on a form. This is useful when developing prototypes but does not scale to production databases.
- Unbound – the grid is populated from a SQL recordset which is demand paged from the server.

A simple example demonstrates how to add an unbound grid to view all companies:

```
Public Sub Form_AfterLoad()  
  
    Set dwGrid = FormObjectsLocal.ShowForm.GetFieldWidget( _  
        "EntityName", "GridFieldName")  
    if dwGrid is nothing then exit sub  
  
    With DataWidgets  
        .InitialiseUnboundGrid (dwGrid), "All Companies"  
        .PopulateGridFromSQLQuery (dwGrid), "Select * From Company"  
    End With  
End Sub
```

Not only can the grid now efficiently process very large recordsets, but it also automatically provides



column sorting: by right mouse clicking on the column header.

**Warning:** Column sorting is very generic so you must obey an important rule when writing your SQL query. This rule relates to fields of type TEXT in SQL Server. TEXT fields cannot be sorted unless cast to a VARCHAR data type. For example the JobTitle column should be written as:

```
SELECT..... Convert(Varchar(64), JobTitle) as JobTitle .... FROM....
```

This will ensure that sorting the JobTitle column will be efficiently processed.

It is normal to add your own style to columns (width, type etc..) and this should be done using `DataWidgets.AddColumnToGrid` before calling `DataWidgets.PopulateGridFromSQLQuery`.

### 39. Refreshing Grids on Other Entity Forms

When a form wishes to fire a button on another entity form which can be virtual or not, perhaps to refresh a grid or view, then this example shows how to click the button programmatically from another form script:

```
Private Sub RefreshBidManagerGrid
    Set frmBidManager = _
        FormFunctions.GetAFormInstance( _
            "frmFormDesignedByUser", True, "", "BidManager")

    If frmBidManager Is Nothing Then exit sub ` Form not open

    Set btn = frmBidManager.GetShowForm.GetButtonFromFunction( _
        "btn", "RefreshBidStagesGrid")

    If btn Is Nothing Then Exit Sub ` Button not found

    btnIndex = btn.FormWidget.Index
    frmBidManager.btn(btnIndex).Value = True
End Sub
```

If the form is a virtual entity, replace "frmFormDesignedByUser" with "frmVirtualFormDesignedByUser" in the GetAFormInstance function.

#### 28 March 2002 Update:

It is now possible to use the frm{EntityName} in order to access forms which is more intuitive. The example below shows how this works:

```
If FormFunctions.isformloaded("frmAdvertisement") Then
    Set f = FormFunctions.GetAFormInstance("frmAdvertisement")
    f.LoadEntityDetails 0
    FormFunctions.AddFormToTabView (f), ""
Else
    Set f = FormFunctions.LoadFormByName("frmAdvertisement", 0, "New Advert")
End If
```

## 40. Running Mail Merges From Script

The mail merging technology was embedded within the actions system and not exposed to script until 5.00.091. This build includes a script friendly interface to easily generate mail merges without requiring any pre-processing of the data sources into the expected format.

A new object is exposed: **MailMerge**, which can be accessed from the script editor. This object exposes a number of properties/methods:

Name	Type	Description
NewDocument	Method	Called to instantiate a new merge document
KeySQL	Property	The SQL which generates the ID's of the keys
Template	Property	The word template .DOT file
AppendDocument	Method	Append a document to collection
SaveAs	Property	Save the merged document as this file name
UserId	Property	Set the user who is running the merge
RunMailMerge	Method	Run the merge and generate the document

The use of this technique is now described by use of an example:

```
Private Sub MailMergeTest
  With MailMerge
    .NewDocument
    .KeySQL = "Select CompanyId, 26531 as ContactId, 99 as RequirementId, " & _
              " 25 as ContractId, " & _
              " 33518 as CandidateId, 4 as AdvertisementId, " & _
              " 3 as ContractorInvoiceId, 1 as TestEntityId" & _
              " From Company Where CompanyId = " & m_CompanyId

    .Template = "\\TriSys2000\ActionTemplates\client mailshot.dot"
    .AppendDocument "D:\CVData\P\Synonyms Test.doc"
    .SaveAs "\\TriSys2000\ActionTemplates\Mailshots\MMTest.rtf"
    .RunMailMerge
  End With
End Sub
```

The **.KeySQL** property accepts a SQL string returning **only** the keys of the underlying entities. In this example, we want to merge the document consisting of the current company (m\_CompanyId), requirement 99, placement 25, candidate 33518, advertisement 4, contractor invoice 3 and test entity 1. This allows you to control the relationships between the records.

The **.Template** property is the full path to the word .dot file.

**.AppendDocument** can be called any number of times to append any number of documents (CVs perhaps) to the end of the merge.

The **.SaveAs** method tells the merge that the resulting merged file will be saved to the specified file name.

The **.RunMailMerge** method starts the process of creating the data source (.tmm file), instantiating word, opening the .dot template and merging to a new document using the data source. If the .SaveAs file is set, it then saves the document. The resulting document is then made visible for viewing.

## 41. Diagnostics and User Activity Logging for Debugging and Audit Trails

For debugging, consider using the [Diagnostics](#) object for debugging as this logs to the diags file in real-time without intrusion and can capture hundreds of messages. Turn diagnostics on either via the *Configure->Diagnostics* menu option, or using the */Diag* command line option.

To use in your code, just use the following syntax:

```
Diagnostics.Diag _  
    "Your message goes here... Concatenate variables, properties etc..."
```

The main advantage to this approach is that you can leave these in production systems and messages are only logged when diagnostics are enabled when attempting to debug a problem encountered on-site.

The .out file produced also contains version + ancilliary information which can help diagnose problems.

Note, that when diagnostics are enabled, all [MessageBox.AppMsg](#) and [AppStopMsg](#) and [QuestionYesNo](#) etc.. are also logged to the .out file.

For logging user activity to SQL Server as an audit trail, use the following:

```
Initialisation.Users.DbInsertUserActivity (Initialisation.UserId), _  
    "The message to be logged", startDate, endDate
```

The startDate and endDate can be set to *Now*, but these can be used to capture the time taken for a particular operation. All records are written to the UserActivity table only if the *Configure->User Activity Logging* menu option is set by administrator.

## 42. Closing Action Forms

It may be necessary to close an action form from script for example when a user is not permitted to run an action. In this case the following method can be called from the `Form_AfterLoad` event in order to stop the form from loading and displaying:

```
FormObjectsLocal.FormRef.FormUnload
```

### 43. Intercepting Shortlisting Events

New scriptable event available from both requirement and candidate search forms which is raised when a contact is added to the shortlist either via the shortlist button or via active searching pop-up window.

```
Public Sub Form_AfterShortlist ( ContactId )
```

This event is called after the contact is added to the underlying tables (CandidateSearchHits, RequirementCandidate, RequirementCandidateAction) but before the associated grids are refreshed. This gives the scripter the opportunity to manipulate the underlying records, or indeed add the record to other unrelated tables (customised lists).

```
' Catch each candidate as they are added to the shortlist so that we can  
' do our own thing with them.  
Public Sub Form_AfterShortlist ( ContactId )  
    MsgBox.appMsg "Script: ContactId: " & _  
        ContactId & " Added to requirement: " & m_RequirementId  
End Sub
```

## 44. Post Code Lookup

5.00.100 added post code database lookup capability to the system and exposed this as script. If the customer has licensed the post code address file (PAF) from QAS Systems, then the post codes will exist in the database.

The object PostCodeLookup is exposed through script and can be used as follows:

```
Public Sub Button_Click ()
    Set btn = _
        FormObjectsLocal.ShowForm.GetButtonFromIndex("btn", _
            FormObjectsLocal.EventControl.Index)
    If btn Is Nothing Then Exit Sub
    On Error Resume Next
    Select Case btn.ButtonFunction
        Case "AlternativeAddressPostCodeLookup"
            Call AlternativeAddressPostCodeLookup
    End Select
End Sub

' Button lookup from alternative address fields.
Private Sub AlternativeAddressPostCodeLookup
    sPostCode = FormObjectsLocal.ShowForm.GetFieldValue( _
        "ContactConfigFields", "AlternativePostCode")
    If len(sPostCode) = 0 Then
        MsgBox.VBMsgBox gVBMsgBoxSpaceOut ( _
            "Please enter a post code before searching for a matching address.")
        Exit Sub
    End If

    ' Do the lookup now
    With PostCodeLookup
        .PostCode = sPostCode
        If .Search Then
            FormObjectsLocal.ShowForm.SetFieldValue("ContactConfigFields", _
                "AlternativeStreetAddress") = .Street
            FormObjectsLocal.ShowForm.SetFieldValue("ContactConfigFields", _
                "AlternativeStreetSuburb") = .Suburb
            FormObjectsLocal.ShowForm.SetFieldValue("ContactConfigFields", _
                "AlternativeCity") = .City
            FormObjectsLocal.ShowForm.SetFieldValue("ContactConfigFields", _
                "AlternativeCounty") = .County
            FormObjectsLocal.ShowForm.SetFieldValue("ContactConfigFields", _
                "AlternativePostCode") = .PostCode
        End If
    End With
End Sub
```

Other properties of the PostCodeLookup object are as follows:

Enabled	Read	True of post code lookup enabled for site
PostCodeRadialSearching	Read	True if radial searching enabled for site
TopRecordCount	Write	Number of records to display in wildcard post code search
PopUpSelection	Write	Set to True if a wildcard pop-up selection dialogue is to be displayed.

## 45. Using Bookmarks in Grids

As a rule, it is not advisable to enumerate through the grid to access the rows and cells directly as this is inefficient. See the following section to see how to access the underlying unbound recordset.

In order to retrieve all rows in a non-additem grid, use the following code:

```
With dwGrid
    .Redraw = False

    .Movelast
    .Movefirst

    For i = 0 to .Rows - 1
        .Bookmark = i
        ContactId = .Columns(0).Text
        If i = 0 Then
            s = s & (ContactId)
        Else
            s = s & ", " & (ContactId)
        End If
    Next

    .Redraw = True
End With
```

## 46. Accessing Grid SQL Query Recordset and Cloning

When a grid is populated from a single SQL statement using the function: `DataWidgets.PopulateGridFromSQLQuery`, the grid handling is performed automatically by the application.

There are occasions however, when you may require access to the underlying recordset in order to perform some extra processing (calculating totals etc..).

The following example shows how to populate a grid, retrieve the automatically generated underlying recordset, clone it and enumerate through the cloned recordset to access the values:

```
With DataWidgets
    .InitialiseUnboundGrid (dwGrid), "First 1000 Rows of Table: " & sTableName

    sSQL = "Select Top 1000 * From " & sTableName

    ' One stop shop - even easier than drinking beer!
    .PopulateGridFromSQLQuery (dwGrid), (sSQL)
End With

' Test access to the recordset using cloning technique
s = ""
Set rs = DataWidgets.GetUnboundScriptedGridRecordsetFromHwnd(dwGrid.hWnd)
If Not rs Is Nothing Then
    Set cloneRS = rs.Clone
    With cloneRS
        .MoveFirst
        Do While Not .EOF
            s = s & " " & .Fields(1).Value & .Fields(2).Value
            .MoveNext
        Loop
    .Close
    End With
End If
MessageBox.appmsg "Recordset contents=" & s
```

## 47. Unbound Grids - Formatting Columns

As regards formatting, there are 2 ways to achieve this: Using SQL (fastest and most efficient) and using Row Level Callbacks (slower, but more control)

### SQL Method

Code the SQL Select with the convert function to get a UK display date format:

```
Select convert(varchar(11), StartDate, 113) as StartDate
```

The numbers might not be exactly what you want but this is what you do to get your formatting:

```
Select Convert(varchar(11), convert(decimal(10,5), numField)) as numField
```

This will give you a number with 5 decimal places. (Notice the double convert to get the value into text format because ADO will strip trailing zeros).

### Row Level Callbacks

Use the Grid\_RowLoaded event to capture the loaded row from the SQL statement and format the specified column explicitly. Example:

```
Public Sub Grid_RowLoaded
    Set dwGrid = FormObjectsLocal.EventControl
    If dwGrid Is Nothing Then Exit Sub
    Set fd = _
        FormObjectsLocal.ShowForm.GetFieldDescriptionFromWidgetHwnd(dwGrid.hWnd)
    If fd Is Nothing Then Exit Sub

    Select Case fd.TableFieldName
        Case "DateViewGrid"
            ' Format the third column (0 base columns) which is a date into
            ' the format e.g. 19-Feb-2002
            With dwGrid
                If .Columns.Count > 2 Then
                    .Columns(2).Text = _
                        StringFunctions.VBFormat(.Columns(2).Text, "DD-MMM-YYYY")
                End If
            End With
        End Select
    End Sub
```

**Warning:** Failure to check the column counter before setting the column value will result in accessing a non-existent grid cell which will cause the grid to bring down the application!

## 48. Fast Contact Search – Script Access

The Selection.FastContactSearch function was added in build 5.00.054 to allow script developer to access the fast contact search pop-up.

An example is shown in section 27.

In order to more tightly control this, a new AND SQL argument has been added in 5.00.121 which can be used by the script programmer to add an extra SQL WHERE clause to the query.

The function signature is:

```
' VB Script friendly function to get a selected contact
Public Function FastContactSelection( _
    ClientContacts As Boolean, _
    CandidateContacts As Boolean, _
    Optional CandidateDefault As Boolean = False, _
    Optional AndSQL As String = "") As Long
```

An example is shown below:

```
Private Sub FastContactSelection
    ' Find a contact - either client (arg1) or candidate (arg2)
    ' - default client (arg3) - called Smith (arg4)
    ContactId = Selection.FastContactSelection(True, True, False, _
        "And Contact.Surname = 'Smith'")
    If ContactId = 0 Then Exit Sub

    Set Contact = Initialisation.Contacts.FindFirst((ContactId))
    MsgBox.AppMsg "You selected contact: " & Contact.Christian & _
        " " & Contact.Surname & "?"
End Sub
```

## 49. Form\_Unload – Preventing form from closing from Script

By using the Form\_Unload (Cancel ) event, it is possible to prevent the user from closing the form without saving the changes controlled by script.

The application knows when a form is 'dirty' i.e. fields have been changed but not saved and it correctly prompts the user in these instances. The application however, knows nothing about the script behaviour and which fields it may control, so offers this event to allow script to prevent the form from closing in such circumstances.

Here is an example:


```
' Ask user to quit
Public Sub Form_Unload ( Cancel )
    Cancel = _
        (MessageBox.QuestionYesNo("Are you sure you want to quit?", _
            "Form_Unload") <> vbYes)

    FunctionStatus.SetParameter("Cancel") = Cancel
End Sub
```

In the above example, the event is called from the application when the user wants to close the form. The event simply asks the user for verification and returns the status to the application. If the user answers Yes, the form will close, otherwise it will not.

## 50. Spin Increments – Setting Visibility from Script

It is sometimes necessary to set the visibility of some fields (Candidate Vs Client etc..) by script.

The numerical data entry fields spin increment widgets:  are separate from the actual field.

They have the same index as the field so it is possible to write a generic function as follows:

```
Public Function gSetSpinIncrementVisibility(sEntityName, sTableFieldName, bVisible)
    With FormObjectsLocal.ShowForm
        Set objField = .GetFieldWidget((sEntityName), (sTableFieldName))
        If objField Is Nothing Then Exit Function
        Set formField = .GetFormFieldFromFieldName((sEntityName),
(sTableFieldName))

        index = objField.Index
        Set spin =
FormObjectsLocal.formref.controls("spinIncrementValue")(index)

        If .IsWidgetVisibleOnCurrentTab((objField.hwnd)) Then
            spin.Visible = bVisible
        End If
    End With
End Function
```

This can be used as follows:

```
Public Sub Tab_Click ()
    Call InvisibleNotClientFields
End Sub

Public Sub Form_AfterLoadRecord ( Id )
    Call InvisibleNotClientFields
End Sub.

Public Sub InvisibleNotClientFields

Call SetFieldVisible("ContactConfigFields", "RequiredRemuneration",
                    ContactType)

Call SetFieldVisible("ContactConfigFields", "RemunerationRequiredPeriod",
                    ContactType)

'set spin increments invisible

Call gSetSpinIncrementVisibility("ContactConfigFields", "RequiredRemuneration",
                                Not ContactTypeClient)

Call gSetSpinIncrementVisibility("ContactConfigFields", "MinimumPerm",
                                Not ContactTypeClient)

End Sub
```

## 51. FreeTextSearchEngine – Searching CV's using ISYS from Script

In build 6.00.089 or later, script developers can now get access to the ISYS CV search functionality.

A new object: **FreeTextSearchEngine** is available for this purpose:

```
` Returns the number of documents found
Public Function Search(sFreeText As String) As Long

` Returns the long path name for the specified index
Public Function GetDocumentLongFileName(Index As Long) As String

` Returns the number of hits in the document at the specified index
Public Function GetDocumentHits(Index As Long) As Long

` Returns the relative path of the document at the specified index
Public Function GetDocumentRelativeFileName(Index As Long) As String
```

Example Code:

```
With FreeTextSearchEngine
    lNumDocuments = .Search("unix // oracle")

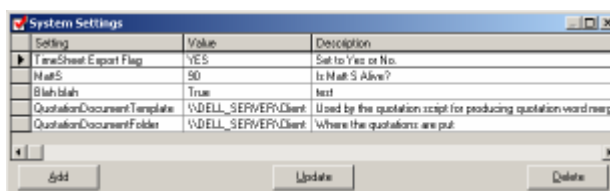
    MsgBox.AppMsg "Found " & lNumDocuments & " CV's?"

    For i = 1 to lNumDocuments
        s = s & vbCrLf & .GetDocumentLongFileName(i)
    Next

    MsgBox.AppMsg "CV's: " & s &
End With
```

## 52. Access System Settings

An administrator can set system wide settings using the Configure→System Settings menu:



To access these from script, use the **Initialisation**.GetSystemSetting(sName) function.

This is available from version 6.01.006.

Example:

```
sSystemSetting = Initialisation.GetSystemSetting ("YourSettingName")
```

Programmers can now generate and update their own settings using the **Initialisation**.SetSystemSetting(sName, sValue) property:

```
Initialisation.SetSystemSetting("GoToAssistDefaultId") = lDefaultId
```

This function writes the new value directly to the database SystemSetting table.

## 53. Appendix A - Type Mismatch Errors

This excerpt from MSDN explains the reason why some functions in TriSys will not be allowed to be called with BYREF parameters because of limitations in the VBScript (or JavaScript) language.

### INFO: Type Mismatch Errors When You Pass Parameters from ASP to a Visual Basic Component

---

The information in this article applies to:

- Active Server Pages
  - Microsoft Visual Basic Learning, Professional, and Enterprise Editions for Windows, versions 5.0, 6.0
- 

#### SUMMARY

The code sample below illustrates various scenarios that causes

##### Type Mismatch

errors with method calls from an Active Server Page (ASP) using Visual Basic Script to Visual Basic COM components.

```
'VB Code : [Project=prjParam;Class=clsParam]
'By default, the parameter is passed 'ByRef'

Sub x( a as string )
    a = "Changed"
End Sub

'ASP Code
k = "Hello"

'Create the above VB object

Set obj = Server.CreateObject("prjParam.clsParam")

obj.x k          'Type Mismatch error occurs

obj.x (k)        'Using PARANTHESIS forces 'ByVal' , 'k' does not change

Call obj.x (k)  'Type Mismatch error occurs

Call obj.x cstr(k) 'The CSTR function returns a string,
                  'the address of the variable (k) is not passed.
                  'The value of 'k' doesn't change
```

Set obj = Nothing

The following is another example that can cause the error:

```
'VB Code : [Project=prjParam;Class=clsParam]
'If you do not specify, by default the parameter is passed 'ByRef'
'Note: Parameter type is VARIANT

Sub y( a as variant )
    a = "Changed"
End Sub

'ASP Code
```

```
k = "Hello"

'Create the above VB object

Set obj = Server.CreateObject("prjParam.clsParam")

obj.y k          'changes 'k'

obj.y (k)        'Using PARANTHESIS forces 'ByVal' , 'k' doesn't change

Call obj.y (k)  'changes 'k'

Set obj = Nothing
```

## MORE INFORMATION

**VBScript** only supports VARIANT **ByRef** parameters. You can use **VBScript** to call a procedure that takes **ByRef** strings, but the default behavior of components built with Visual Basic is to fail with a type mismatch error when trying to pass **ByRef** parameters to these components. OLE Automation's default type-coercion function fails when asked to convert a **ByRef** variant into any other **ByRef** type.

**VBScript** does not impose this restriction. However, it is the default behavior of the component that decides that a **ByRef** variant cannot be converted into a **ByRef** string.

If a parameter to a procedure is enclosed in parenthesis, the parameter is first evaluated as an expression. Because the result of an expression cannot be passed by reference, the parameter is passed *ByVal* and no error is reported.

Avoid using **ByRef** parameters unless they are explicitly needed. This is because:

- **ByRef** procedures cause more overhead during cross-process marshaling, because COM must marshal the value both to and from the object. *ByVal* parameters require only one-way marshaling.

However, if you are willing to accept the overhead of any marshaling at all, an extra parameter marshal back is unlikely to cause a huge performance degradation.

- **ByRef** parameters can introduce hard-to-find bugs in your code if you accidentally change the value of one of the parameters.
- JScript does not support **ByRef** parameters of any type, so if you plan to write components that will support JScript, you should not use **ByRef** parameters at all.