finPOWER Connect 3
Page Sets

Version 3.04

18th May 2017

Table of Contents

Disclaimer

This document contains information that may be subject to change at any stage.

All code examples are provided "as is".

This document may reference future functionality not currently available in the release version of finPOWER Connect.

Version History

Date	Version	Name	Changes
25/06/2014	1.00	PH	Created.
26/06/2014	1.01	PH	Updated after JR review.
15/07/2014	1.02	PH	Updated for changes made in build 2.01.01.00 + FAQ sections.
05/08/2014	1.03	PH	New functionality and information, e.g., executing an Action Script and Query and Report details.
05/08/2014	1.04	PH	Disclaimer updated and Page Set Events section added.
06/08/2014	1.05	PH	Updated for 'Inline Tabs' type Page Sets.
13/08/2014	1.06	PH	Notification actions clarified.
19/08/2014	1.07	PH	Added ExectionAction methods to Page Set Handler.
29/08/2014	1.08	PH	FormHeadingColour added for v2.02.00.
23/09/2014	1.09	PH	GetCodeDescriptionListFromEnquiryAction documented.
20/10/2014	1.10	PH	Examples of using flags and colour blocks in grid columns and clearing grids.
18/11/2014	1.11	PH	Grid events and selected vs active rows clarified plus other tweaks.
26/11/2014	1.12	PH	New FormShowCompanyLookup method.
22/12/2014	1.13	PH	Updated for new HTML Panel and Grid functionality.
07/01/2015	1.14	PH	Icon enhancements, e.g., when setting grid cell value.
12/01/2015	1.15	PH	PageSetActivate event enhancements.
13/03/2015	1.16	PH	Account Application special forms detailed.
07/04/2015	1.17	PH	Updated Image Page Object type to show Icon code.
05/05/2015	1.18	PH	Updated for new grid save/ load layout methods.
23/07/2015	1.19	PH	Updated for new FormShowSecurityStmtItem method.
14/08/2015	1.20	PH	Added HTML Editor type Page Objects.
14/09/2015	1.21	PH	HTML Editor now has an option to follow hyperlinks when read-only and Number Boxes now have an Allow Blank property.
28/09/2015	1.22	PH	Added clarification of how to access the Parent Form from a Page Set when opened modal/ pseudo-modal from an Application Shortcut.
16/10/2015	1.23	PH	Added StringListCsv type columns.
25/01/2016	1.24	PH	Added FormShowAccountTemp and FormShowClientTemp methods and grid row Visible property. Plus support for Credit Enquiry and PPSR Search forms.
12/07/2016	3.00	PH	Updated for finPOWER Connect version 3.
07/11/2016	3.01	PH	Enter and Leave events added.
17/01/2017	3.02	PH	Added 'DateTimeZone' Page Object.
24/03/2017	3.03	PH	FormShowAccountPaymentArrangementAdd method added.
18/05/2017	**3.04**	**JR**	**Clarified Change Event not fired when Value changed in Code.**

Introduction

This document discusses finPOWER Connect Page Sets and is focused on the creation and configuration of Page Sets.

Page Sets require that finPOWER Connect is licensed for the Page Sets Add-On.

If a Page Set is being created for use with Account Applications, this document should be used in conjunction with the **finPOWER Connect 3 Account Applications** document.

Samples

Several Page Set samples are available. Each of these is documented in Appendix C – Samples.

The sample Page Sets are included within the finPOWER Connect setup and are stored in the **[ApplicationFolder]/Templates/SampleScripts** folder.

NOTE: Any samples relating to Account Applications are detailed in the **finPOWER Connect 3 Account Applications** document.

Styles and Formatting

Details on styling and formatting Page Sets so that they look similar to built-in finPOWER Connect forms is given in Appendix A – Guidelines.

Security

When querying the finPOWER Connect database, always ensure that your SQL WHERE clause contains any filters relevant to the current User, e.g.:

```
sqb.sqlWhere.Append(finBL.CurrentUserInformation.FilterClientSqlWhere)
```

Variations on this exist for Accounts, Account Applications and Security Statements.

Page Sets Overview

- Page Sets allow custom User Interface forms to be created in finPOWER Connect.
  - These forms can be either wizards, tabbed or single page forms.
  - Like Windows forms, they are event driven and thus provide far more control over the interface than other customisation mechanisms such as Parameter Sets.
- Page Sets rely heavily on Scripting.
  - Page Set Scripts have access to the full finPOWER Connect business layer functionality.
- Page Sets are Windows based and there is no concept of a Page Set running in a Web environment.
- Page Sets can contain one or more pages of information.
  - Pages contain Page Objects such as Text Boxes, Grids, HTML Panels etc.
  - Pages can be set to 'Flow' layout which positions the Page Objects automatically. This allows pages to be created much more quickly than 'Positioned' layout where Page Objects are individually positioned on the page.

Page Sets Form

The various pages on the Page Sets form are described in this section. Many properties of Page Sets are expanded on in later sections.

General

General details.

- **Code and Description**
  - Code:
    - A unique code up to 10 characters long.
  - Active:
    - Inactive Page Sets may still be run however, they do not show when creating new records, e.g., in the Page Set dropdowns on the Account Application Types form.
  - Description:
    - A description of the Page Set, up to 50 characters long.
- **Beta Status**
  - You can flag a Page Set as being Beta, e.g., still undergoing testing. This will append the text '(Beta)' to the form title when running the Page Set.
- **Notes**
  - Notes regarding the Page Set.

Options

Miscellaneous options. These mainly relate to how the Page Set form will look when the Page Set is run.

- **Form Type**
  - Form Type:
    - Page Sets can be one of three form types:
      - Single Page
      - Tabbed Pages
      - Wizard
      - Inline Tabs
    - These are described in detail in the Form Types section.
  - **Minimum Form size**
    - Width:
      - The minimum width to size the Page Set form. If unspecified, 400 pixels (px) is assumed.
    - Height:
      - The minimum height to size the Page Set form. If unspecified, 400px is assumed.
  - **Form Heading and Title**
    - Show Form Heading?
      - Indicates whether to display a Form Heading region. The Form Heading is the white block containing a bold title and a smaller summary underneath and appears on most forms within finPOWER Connect.

        Each Page in the Page Set can define a Heading and Summary which are displayed in the Form Heading region.
    - Form Title:

        - By default, the title of the Page Set form is set to the Page Set's Description. This can be overridden here.
- **Command buttons**
    - Show 'OK' button?
        - Indicates whether to display an 'OK' button on the form.

          NOTE: Not applicable to Form Type 'Wizard'.
        - Caption:
            - The caption to display on the 'OK' button.

              NOTE: This will, by default, be 'Finish' rather than 'OK' if the Form Type is 'Wizard'.
    - Show 'Cancel' button?
        - Indicates whether to display a 'Cancel' button on the form.

          NOTE: Not applicable to Form Type 'Wizard'.
        - Caption:
            - The caption to display on the 'Cancel' button.
    - Show 'Save' button?
        - Indicates whether to display a 'Save' button on the form.
        - Caption:
            - The caption to display on the 'Save' button.
    - Show 'Print' button?
        - Indicates whether to display a 'Print' button on the form.
        - Caption:
            - The caption to display on the ' Print' button.

Pages

Displays a list of Pages defined for the Page Set.

A preview of the Page selected in the Grid is shown on the right and a summary of the Page is shown below.

Pages can be added, deleted, edited and re-ordered.

Editing of Pages is described later in this document.

Script Code

Page Sets use Script code for almost everything. It is not possible to have a functional Page Set without at least some Script code running behind the scenes.

Unlike many other Scriptable objects (Scripts, Documents, Queues etc.), a Page Set's Script does not have a Main method. The Script object is created when the Page Set is run and is therefore 'stateful', i.e., any member variables defined within the Script code will retain their values between calls. This is unlike most finPOWER Connect Scripts which are 'stateless', i.e., the Script object is created, the Main method called and then the Script object destroyed.

NOTE: Right-clicking a Page Object (or the Page background) on the preview on the Pages page shows a list of events. Selecting one of these events will either insert or go to an existing event handler in the Script. Events are shown in blue in the context menu if they already have a handler in the Script code.

Constants

Like any other admin library that uses Scripts, Constants can be defined on this page and used by the Script.

Usage

Displays a list of places where this Page Set is used within finPOWER Connect, e.g., where the Page Set is used by an Account Application Type.

NOTE: This page only shows where there is a 'direct' usage of the Page Set, e.g., between Admin Libraries. Referencing a Page Set elsewhere, e.g., from a Task Manager Folder, will not show on this page.

Form Types

Page Sets can be configured to use one of three different form types as well as the special 'Inline Tabs'. These are described in this section.

NOTE: Under certain circumstances, the Form Type defined on the Options page of the Page Sets form may be overridden, e.g., an Account Application may display a 'Wizard' type Page Set for initial data capture but may revert to a 'Tabbed Pages' form once data entry has been completed.

Wizard

Wizard Page Sets should be used where data entry needs to be completed in a linear fashion and where branching may occur. E.g., if entering Client details, you may wish to display a different page depending on whether the Client is an Individual or Company.

By default, a Wizard will move though all Active pages defined on the Pages page in sequence.

A wizard has **Cancel**, **< Back**, **Next >** and **Finish** buttons to allow navigation and works as follows:

- When the wizard is first displayed and when moving forward to the next page (using the **Next >** button), the **WizardMove** event is called.
  - The Script can then cause a page to be skipped by setting `e.ShowPage = False`.
- The **WizardRefresh** event is called when the page is first displayed and when moving forward through the wizard.
  - This is NOT called when moving backwards.
  - Allows the Script to load information for the Page Objects on this page if necessary or update the state of Page Objects.
- When moving forward through the wizard, the **WizardValidate** event is called.
  - This is NOT called when moving backwards.
  - Allows the Script to validate the data entered on the current page and prevent the User moving to the next page.
  - The Script indicates that a page is not valid by setting `e.Failed = True`.
- When a page is displayed, the **WizardButtonsUpdate** event is called.
  - Allows the Script to tweak the wizard buttons, e.g., disable the **Next >** button and enable the **Finish** button if the Script detects that the User is on the last page of the wizard (but it is not really the last page).
- When the **Finish** button is clicked, the **WizardValidate** event is first called and, providing this is successful, the **CommandButtonClick** event will be called.

Event handlers can be added to the Script by right-clicking the Page background when in edit mode. These events are described below.

WizardMove

This event allows the Script to decide whether or not to display a particular wizard page.

Inserting an event handler will create a sample event handler Script method which contains a `Select Case` block for each page in the Page Set.

Setting `e.ShowPage = False` will cause the page to be skipped, e.g.:

```
Public Sub PageSet_WizardMove(sender As Object,
                              e As finPageSetHandlerWizardMoveEventArgs) Handles Me.WizardMove

  Select Case e.PageId
    Case "ClientType"
    Case "Individual"
      If chkClientIsCompany.Value Then e.ShowPage = False
```

```
    Case "Company"
      If Not chkClientIsCompany.Value Then e.ShowPage = False
  End Select

End Sub
```

WizardRefresh

This event allows the Script to perform some action when a page is moved forward to, e.g., populate Page Objects, bind a collection to a grid or update the state of Page Objects based on information entered on an earlier page.

Inserting an event handler will create a sample event handler Script method which contains a `Select Case` block for each page in the Page Set.

WizardButtonsUpdate

By default, the **< Back**, **Next >** and **Finish** buttons will be enabled and disabled automatically, e.g., if the User is on the first page of the wizard, the **< Back** button will be disabled; if they are on the last page, the **Finish** button will be enabled and the **Next >** button disabled.

There may be occasions where you may wish to override this functionality, e.g.:

- A wizard that you can finish from any page rather than just the last page.
- A wizard where the last page will never be shown due to some earlier choice. In this case, you may wish to enable the **Finish** button and disable the **Next >** button.

The following sample updates the state of the **Finish** and **Next >** buttons based on the current wizard Page:

```
Public Sub PageSet_WizardButtonsUpdate(sender As Object,
                                       e As finPageSetHandlerWizardButtonsUpdateEventArgs) Handles
Me.WizardButtonsUpdate

  e.FinishEnabled = (e.PageId = "Individual" OrElse e.PageId = "Company")
  e.NextEnabled = Not e.FinishEnabled

End Sub
```

NOTE: The `e` object properties relating to standard wizard buttons will always be set to their default states every time this event is called, e.g., `e.FinishEnabled` will always be `True` on the last page of the wizard but `False` for all other pages. These values can then be overridden if required; although, attempting to enable the **Next >** button on the last page of the wizard or the **< Back** button on the first page of the wizard will have no effect.

Although command buttons can be updated via the `psh.CommandButtons` collection, it is advisable to only use the WizardButtonsUpdate event for wizard type Page Sets.

If you need to force the WizardButtonsUpdate event to fire from your Script, e.g., use the `psh.PerformWizardButtonsUpdate()` method.

The following sample has CheckBoxes to enable and disable the **Next >** and **Print** buttons:

```
Public Sub chkEnableNext_Change(sender As Object,
                                e As finPageObjectChangeEventArgs) Handles chkEnableNext.Change

  ' Refresh wizard buttons
  psh.PerformWizardButtonsUpdate()

End Sub

Public Sub chkEnablePrint_Change(sender As Object,
                                 e As finPageObjectChangeEventArgs) Handles chkEnablePrint.Change

  ' Refresh wizard buttons
  psh.PerformWizardButtonsUpdate()

End Sub
```

```
Public Sub PageSet_WizardButtonsUpdate(sender As Object,
                                       e As finPageSetHandlerWizardButtonsUpdateEventArgs) Handles
Me.WizardButtonsUpdate

  If chkEnableNext.Value Then
    ' Normal behavior, i.e., enable Next from the first page
  Else
    ' Act like a single-page wizard, i.e., disable Next and enable Finish from the first page
    e.NextEnabled = False
    e.FinishEnabled = True
  End If

  ' Other buttons
  e.PrintEnabled = chkEnablePrint.Value

End Sub
```

WizardValidate

This event allows the Script to validate the contents of the current page and prevent the user from moving to the next page if necessary.

Inserting an event handler will create a sample event handler Script method which contains a `Select Case` block for each page in the Page Set.

NOTE: This event is only called when moving forward through the wizard and is only called if all Page Objects on the page are in a valid state, e.g., any mandatory Page Objects have a value entered.

The following example shows how to prevent the User from moving to the next page and also to prompt the User before moving to the next page:

```
Public Sub PageSet_WizardValidate(sender As Object,
                                  e As finPageSetHandlerWizardValidateEventArgs) Handles
Me.WizardValidate

  Select Case e.PageId
    Case "ClientType"
      If MsgBox("Are you sure you have selected the correct Client Type?", MsgBoxStyle.Question Or
MsgBoxStyle.YesNo) = MsgBoxResult.No Then
        e.Failed = True
      End If

    Case "Individual"
      If Len(txtFirstName.Text) = 0 AndAlso Len(txtLastName.Text) = 0 Then
        e.Failed = True
        mUI.MsgBox("You must enter either the First or Last Name.", MsgBoxStyle.Exclamation)
        txtFirstName.Focus()
      End If

    Case "Company"
  End Select

End Sub
```

CommandButtonClick

This event is called when the **Finish** button is clicked (or the **Save** and **Print** buttons if the Page Set is configured to show these) and allows the Page Set Script to perform whatever action is required.

Inserting an event handler will create a sample event handler which contains a `Select Case` block for each command button.

By default, the Page Set will automatically be closed upon clicking the **Finish** button. This can be prevent by setting `e.Cancel = True`.

The following example simply displays a message upon clicking the **Finish** button and then closes the Page Set form. It also prompts to User to close the form if they click the Window Close button:

```
Public Sub PageSet_CommandButtonClick(sender As Object,
                                      e As finPageSetHandlerCommandButtonClickEventArgs) Handles
Me.CommandButtonClick

  Select Case e.CommandButton
    Case isefinPageSetCommandButton.Finish
      If MsgBox("Finish and close this Page Set?", MsgBoxStyle.Question Or MsgBoxStyle.YesNo) =
MsgBoxResult.Yes Then
        ' Perform Finish ations
      Else
        e.Cancel = True
      End If

    Case isefinPageSetCommandButton.WindowClose
      If MsgBox("Close this Page Set?", MsgBoxStyle.Question Or MsgBoxStyle.YesNo) =
MsgBoxResult.No Then
        e.Cancel = True
      End If
  End Select

End Sub
```

NOTE: This event will not fire when clicking the **Next >** or **Finish** buttons if the WizardValidate event indicates that the Page is not valid or if one or more visible Page Objects are not in a valid state.

Tabbed Pages

Tabbed Pages should be used where data entry can be completed in any order or if you need to present multiple pages of information to the User.

By default, Tabbed Pages display **OK** and **Cancel** buttons but this can be customised on the Options page of the Page Set. If neither button is displayed, the Page Set can still be closed by clicking the Window Close button.

Tabbed Pages work as follows:

- The Page Set Script's Initialise method can decide what pages to display.
- When the **OK** or **Cancel** buttons are clicked, the **CommandButtonClick** event is called.

Initialise

This method can be used to hide certain pages as per the following example:

```
Private mClient As finClient

Public Overrides Function Initialise() As Boolean

  ' Assume Success
  Initialise = True

  ' Initialise
  mReports = DirectCast(psh.Reports, ISfinReports)
  mUI = DirectCast(psh.UserInterface, ISUserInterfaceBL)

  ' Load Client
  mClient = finBL.CreateClient()
  If mClient.Load("C10000") Then
    psh.Pages("Individual").Visible = mClient.IsIndividual
    psh.Pages("Company").Visible = mClient.IsOrganisation
  Else
    Initialise = False
  End If

End Function
```

Command Buttons

Command Buttons appear at the button of the Page Set form and, which buttons are displayed is configured on the Page Set form, Options page.

The Page Set Handler (psh) has a `CommandButtons` collection that can be used to update the states of these buttons.

This collection contains an entry for each type of Command Button, regardless of whether the button is applicable (E.g., it will always contain an `isefinPageObjectCommandButton.Finish` entry, even for non-wizard Page Sets). Updating a Command Button that is not applicable to a particular type of Page Set form will have no effect.

Command Buttons have the following properties which can be set by the Page Set Script:

- Caption
  - The button caption.
  - Command Buttons are fixed at 80 pixels wide so short captions should be used.
- Enabled
  - Allows the Command Button to be enabled or disabled.
- Visible
  - Allows the Command Button to be hidden.

The following example updates Command Button states:

```
With psh.CommandButtons(isefinPageSetCommandButton.Cancel)
  .Enabled = Not .Enabled
```

```
  .Caption = "Cancel Edit"
End With

With psh.CommandButtons(isefinPageSetCommandButton.Print)
  .Visible = Not .Visible
End With
```

NOTE: Typically, wizard type Page Sets would not access the `CommandButtons` collection, they use the WizardButtonsUpdate event to update command button states.

CommandButtonClick

This event is called when the **OK** or **Cancel** buttons are clicked (or the **Save** and **Print** buttons if the Page Set is configured to show these) and allows the Page Set Script to perform whatever action is required.

Inserting an event handler will create a sample event handler which contains a `Select Case` block for each command button.

By default, the Page Set will automatically be closed upon clicking the **OK**, **Finish** and **Cancel** buttons. This can be prevented by setting `e.Cancel = True`.

The following example displays a message upon clicking the **Finish** button and then closes the Page Set form. It also prompts to User to close the form if they click the Window Close button:

```
Public Sub PageSet_CommandButtonClick(sender As Object,
                                      e As finPageSetHandlerCommandButtonClickEventArgs) Handles
Me.CommandButtonClick

  Select Case e.CommandButton
    Case isefinPageSetCommandButton.Ok
      If MsgBox("Finish and close this Page Set?", MsgBoxStyle.Question Or MsgBoxStyle.YesNo) =
MsgBoxResult.Yes Then
        ' Perform OK ations
      Else
        e.Cancel = True
      End If

    Case isefinPageSetCommandButton.Cancel, isefinPageSetCommandButton.WindowClose
      If MsgBox("Close this Page Set?", MsgBoxStyle.Question Or MsgBoxStyle.YesNo) =
MsgBoxResult.No Then
        e.Cancel = True
      End If
  End Select

End Sub
```

NOTE: This event will not fire when clicking the **OK** button if one or more visible Page Objects are not in a valid state, e.g., a mandatory Page Object does not have a value entered.

CurrentPageChanged

This event allows the Script to perform some action when the current page changes, e.g., to load information to be displayed on the page on-demand or to update Page Objects based on information entered on an earlier page.

Inserting an event handler will create a sample event handler Script method which contains a `Select Case` block for each page in the Page Set.

NOTE: The first time this event is called (when the form first loads), both the `e.PageId` and `e.PreviousPageId` properties will point to the same page.

Single Page

Single Pages should be used where only one page of information is to be captured or displayed to the User. They work in exactly the same way as Tabbed Pages type Page Sets.

A Single Page type Page Set can contain multiple pages but it is up to the Page Set Script itself to switch pages (by setting the Page Set Handler's `CurrentPageId` property), e.g., when a button is clicked as per the following example:

```
Public Sub cmdShowMore_Click(sender As Object,
                             e As finPageObjectClickEventArgs) Handles cmdShowMore.Click

  psh.CurrentPageId = "Individual"

End Sub
```

Inline Tabs

Page Sets configured to use 'Inline Tabs' differ from other Page Sets in that they are not intended to be displayed using their own, standalone, form but rather to reside within another form.

Inline Tabs work as follows:

- The Page Set Script's Initialise method can decide what pages to display.
- Since there are no command buttons for 'Inline Tabs', Page Set specific events are limited to the following:
  - PageSetActivate
    - This occurs every time the Page Set is activated, e.g., from the Task Manager, this occurs every time the User switches to the corresponding Task Manager folder.
  - PageResize

Inline Tabs are supported in the following places:

- **Task Manager**
  - Task Manager folders can be defined as 'Page Set' type folders.
  - When the folder is selected, the Page Set will be displayed, allowing the User to interact within it.
  - This is ideal for functionality such as interactive dashboards or reports.
    - Prior to being able to use Page Sets, 'HTML Report' type folders were the only way to provide an interactive environment from the Task Manager.

Pages

A Page Set can contain one or more Pages.

Pages are maintained via the 'Pages' page of the Page Sets form. This shows a list of pages from which Pages can be added, edited, deleted and re-ordered.

Adding a new Page or drilling down to an existing Page displays the Page Wizard.

Page Wizard

This wizard has two pages as described below.

Page

- Each Page must be given a unique code up to 25 characters long. This allows the Page to be identified in the Page Set's Script.
- The Title and Summary of a Page will be displayed in the Form Heading when running the Page Set providing that the Page Set is configured (via the Options page) to display a Form Heading.
    - For 'Tabbed Pages' type Page Sets, the Title is also the caption that is displayed on the Tab representing the Page.
- Layout Mode determines whether the Page's content (Page Objects) are positioned automatically (Flow) or manually (Positioned).
    - These are described in the Page Layout Modes section.
- If a Page's Active flag is unchecked, the Page will not be included in the Page Set and any attempt to reference it via the Page Set Script will result in an error.

Page Designer

The Page Designer page is where the Page's layout is created.

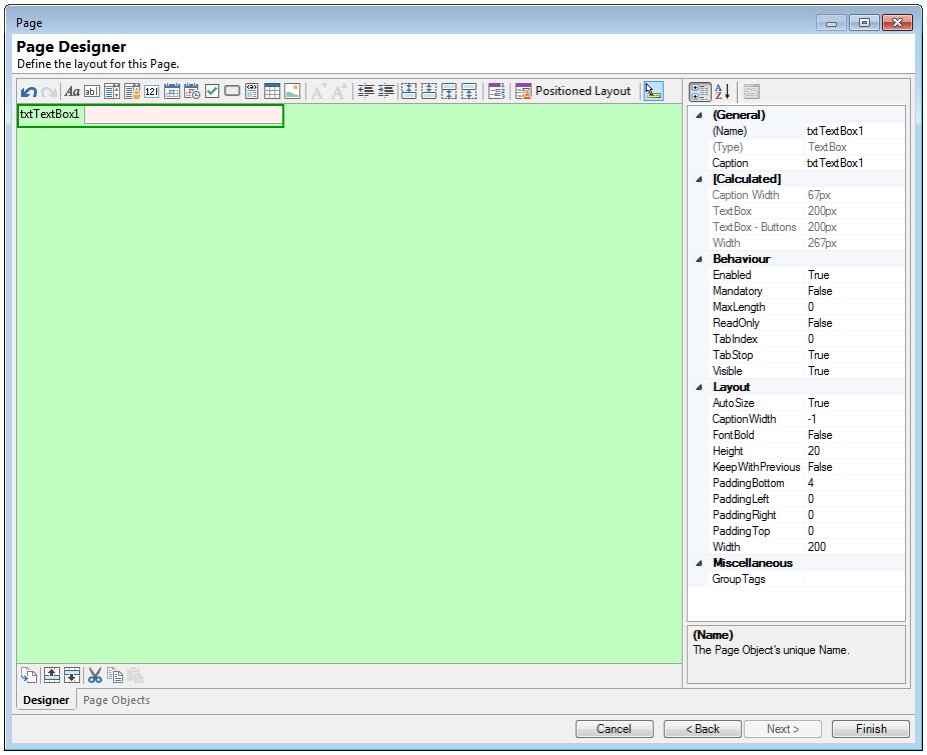
The Page Designer consists of two tabs:

- **Designer**
    - This shows a preview of the page and allows Page Objects to be added, updated, deleted and moved.
- **Page Objects**
    - A grid of Page Objects. This is useful for re-ordering Page Objects or locating Page Objects that, for some reason, cannot be seen on the Designer tab.
    - Page Objects can be added, updated, deleted and moved using the buttons below the grid.

The Page Designer is covered in the next section.

Page Designer

The following screenshot shows the Page Designer which is accessed from the Page wizard:



The Page Designer consists of the following:

- Two tabs; Designer and Page Objects.
  - The Page Objects tab shows a grid of Page Objects. This is useful for re-ordering Page Objects or locating Page Objects that, for some reason, cannot be seen on the Designer tab.
- A toolbar at the top.
- A 'canvas' (the green area).
  - This shows how the Page will look and Page Objects can be selected and moved about.
- A properties grid to the right of the canvas.
  - This allows common properties to be changed for the selected Page Object.
  - If multiple Page Objects are selected, this only allows the properties common to each of the selected Page Objects to be changed.
- A toolbar below the canvas.
  - This allows simple actions based on the selected Page Objects such as duplicating and copying and pasting.

Toolbar

The buttons available on the Page Designer toolbar depend on whether the Page is in 'Flow' or 'Positioned' layout mode.

The table below describes each button.

	Undo	Undo last change.
	Redo	Redo last undone change.
	Label	Insert a Label. You may also drag the button to canvas.
	TextBox	Insert a TextBox. You may also drag the button to canvas.
	ComboBox	Insert a ComboBox. You may also drag the button to canvas.
	DBComboBox	Insert a DBComboBox. You may also drag the button to canvas.
	NumberBox	Insert a NumberBox. You may also drag the button to canvas.
	DateBox	Insert a DateBox. You may also drag the button to canvas.
	Date Cycle ComboBox	Insert a Date Cycle ComboBox. You may also drag the button to canvas.
	CheckBox	Insert a CheckBox. You may also drag the button to canvas.
	Button	Insert a Button. You may also drag the button to canvas.
	HTML Panel	Insert an HTML Panel. You may also drag the button to canvas.
	Grid	Insert a Grid. You may also drag the button to canvas.
	Image	Insert an Image. You may also drag the button to canvas.
	Decrease Label Font Size	Decrease the size of the selected Label(s) font.
	Increase Label Font Size	Increase the size of the selected Label(s) font.
	Decrease Indent	Decrease the selected Page Objects' indent by 8. Flow layout mode only.
	Increase Indent	Increase the selected Page Objects' indent by 8.

		Flow layout mode only.
	Decrease Top Padding	Decrease the selected Page Objects' top padding by 4. Flow layout mode only.
	Increase Top Padding	Increase the selected Page Objects' top padding by 4. Flow layout mode only.
	Decrease Bottom Padding	Decrease the selected Page Objects' bottom padding by 4. Flow layout mode only.
	Increase Bottom Padding	Increate the selected Page Objects' bottom padding by 4. Flow layout mode only.
	Tab Order Mode	Enter/ Exit Tab Order mode.
	Positioned Layout	Change this Page's layout mode to 'Positioned' but retain the current layout. Flow layout mode only.

Canvas

The canvas is where Page Objects are positioned and shows a preview of how the Page will look.

- In 'Flow' layout mode, the Ctrl+Up and Ctrl+Down keys will move the selected Page Object(s) up or down.
  - The toolbar buttons beneath the canvas can also be used to move Page Objects up and down.
- Multiple Page Objects can be selected by:
  - Clicking on the canvas background and dragging the selection rectangle.
  - Holding down the Ctrl key when clicking a Page Object.
    - This will deselect a Page Object if it is already selected.
  - Selecting the Page Object rows in the grid on the Page Objects tab and then switching back to the Designer tab.
- Pressing the Delete key will delete all selected Page Objects.
- Double-clicking a Page Object or right-clicking and selecting 'Properties…' displays the Page Object wizard allowing any of the Page Object's properties to be changed.

The toolbar beneath the canvas has the following buttons:

	Duplicate	Duplicate the active Page Object.
	Move Up	Moved selected Page Objects up. Flow layout mode only.
	Move Down	Moved selected Page Objects down. Flow layout mode only.
	Cut	Cut the selected Page Objects to the clipboard. **NOTE:** Page Objects can be copied between Pages and Page Sets.
	Copy	Copy the selected Page Objects to the clipboard. **NOTE:** Page Objects can be copied between Pages and Page Sets.
	Paste	Paste Page Objects from the clipboard.

Right-clicking a Page Object will provide the following actions in addition to the standard Cut/ Copy/ Paste actions:

	Properties Show the Page Object wizard.
	Copy assignments To Copy a code snippet to the clipboard to assign a value to a Page Object (or multiple Page Objects if more than one is selected), e.g., `txtTextBox1.Text = ""`
	Copy assignments From Copy a code snippet to the clipboard to get a value from Page Object (or multiple Page Objects if more than one is selected), e.g., `value = txtTextBox1.Text`

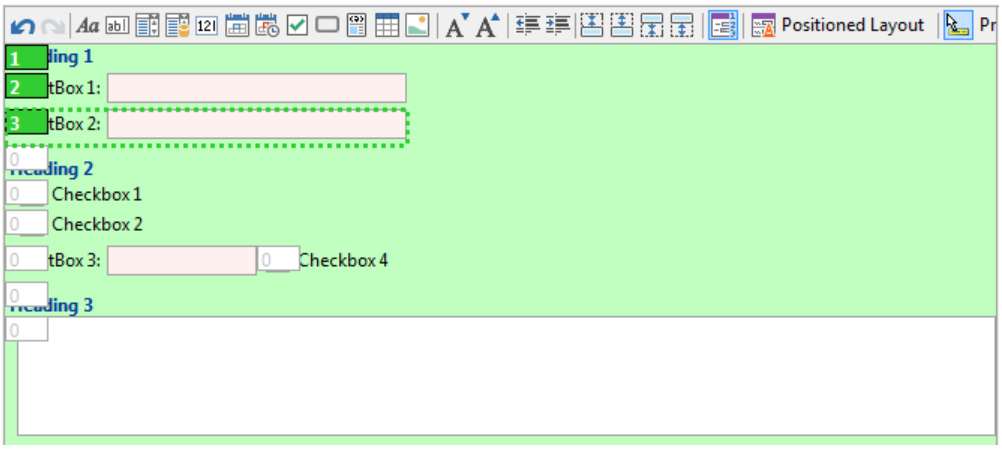
	Delete the selected Page Objects.

Tab Order Mode

By default, all Page Objects have a Tab Index of 0 meaning that, when the User presses the tab key (or Enter key if configured under User Preferences) to move to the next Page Object, the focus will be moved according to the Page Object's order in the Page Objects grid of the Page Designer.

When the Tab Order mode button () is clicked, the Page Designer enters Tab Order mode.

When in this mode, you can click each Page Object in the order you want them to be focused. A number is shown in the top-left of the Page Object denoting the order in which they have been clicked, e.g.:



Tab Order can be accepted by pressing Enter or by clicking the Tab Order mode button. It can be cancelled by pressing the Escape key.

Page Layout Modes

It is recommended that all Pages use 'Flow' layout mode unless there is a good reason not to since flow layout has many advantages as described in the next sections.

Flow

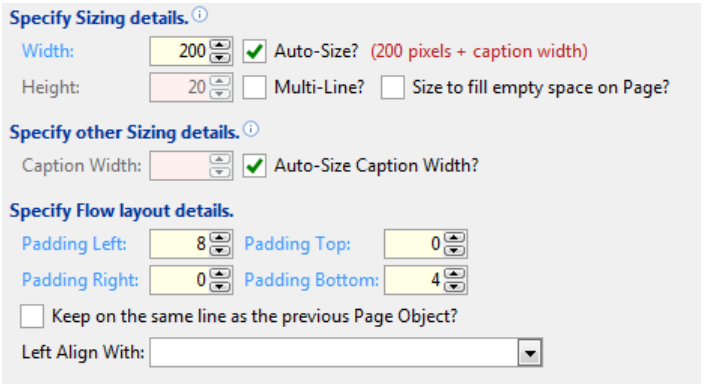
Flow layout mode causes all of a Page's Page Objects to be laid out automatically based on a few simple rules.

This has the following advantages:

- If you hide a Page Object by setting `PageObject.Visible = False`, any Page Objects below it will automatically adjust their positions to fill in the gap where the now invisible Page Object was previously displayed.
- Caption widths can automatically be calculated, e.g., you may add several Page Objects to a Page with differing captions, e.g., First Name, Last Name, Age. By automatically sizing the caption width, you do not have to worry about resizing the Page Objects if you decide to change one of the captions (E.g., changing 'Age' to 'Client's Age').
- A Page Object, e.g., a Grid, can be configured to span the entire Page width so that if the Page is resized, the Grid will grow or shrink to fit.
- A Page Object, e.g., a Grid, can be configured to span the entire Page height so that if the Page is resized, the Grid will grow or shrink to fit the page.
    - If multiple Page Objects are configured for their height to 'Size to fill empty space on page', the remaining white-space at the bottom of the Page will be apportioned equally between these Page Objects.
- Buttons and Labels can size themselves to fit the specified caption, even if the caption is changed from the Page Set Script.

Flow Layout Properties

When viewing a Page Object's properties for a Page that is in 'Flow' layout mode, certain flow-related properties are available:



- **Width**
  The width of the Page Object in pixels. This is actually the width you would like the Layout Rectangle to be (the Layout Rectangle is explained in the next section).
  - **Auto-Size?**
    Auto-sizing the width may have a different effect based on the Page Object type. The red caption to the right explains what it does.

    In this case, it extends the width of the Page Object to include the auto-calculated (in this case) Caption Width. Therefore, if the Caption Width is calculated to be 60 pixels, the actual width of the Page Object's Layout Rectangle will be 260.
- **Height**
  Many Page Objects have a fixed height of 20. In this case, this is a TextBox which, if Multi-Line is checked can vary its height.
  - **Multi-Line?**
    Checking this allows a TextBox type Page Object's height to be specified.
  - **Size to fill empty space on Page?**
    Checking this will cause a TextBox, Grid, HTML Panel or Image type Page Object to expand its height to fill any empty space on the Page. As the Page is resized, the height will be recalculated.

    If a height is specified, this will be taken as being the minimum height for the Page Object.

    If more than one Page Object has this checked, the Page's empty space will be apportioned equally to each Page Object.
- **Caption Width**
  The width of the caption portion of the Page Object in pixels.

  Not all Page Objects have a caption, e.g., Grids and Images.
  - **Auto-Size Caption Width?**
    Checking this will calculate the Caption Width automatically. This is calculated to be the width of the widest caption for all Page Objects on the Page.

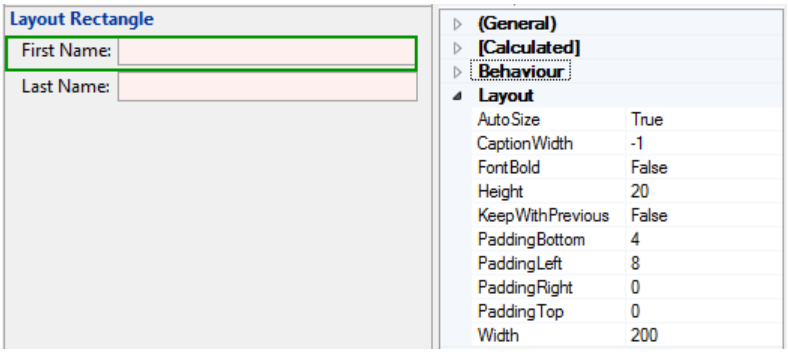
    Where multiple columns are used on the Page, this will be the width of widest caption in this Page Object's column.

    If the Page Object is being kept on the same line as another Page Object, the Caption Width will be calculated based on this Page Object's caption only.
- **Padding Left, Top, Right and Bottom**
  Each Page Object has a Layout Rectangle. The padding settings allow the Page Object to be aligned within this rectangle.

- **Keep on the same line as the previous Page Object?**
  Sometimes it is desirable to have Page Objects appear on the same line on the page, e.g., you may wish to keep several buttons on the same line or maybe the First Name and Last Name of a Client.
- **Left Align With**
  It is possible to left-align a Page Object with another Page Object on the page. This allows you to specify the Name of the Page Object to left-align this Page Object with.

The Layout Rectangle

Page Objects are laid out based upon a layout rectangle. This is simply a rectangle in which the Page Object is positioned but with padding applied to the left, top, right and bottom, e.g.:



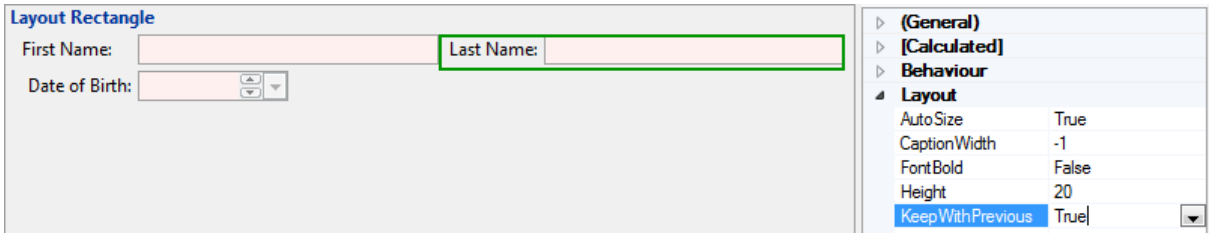
Each of the 3 Page Objects in the above screenshot exists within their own rectangle which is aligned to the left of the Page.

However, both the 'First Name' and 'Last Name' Page Objects have a left padding of 8 and a bottom padding of 4 to space them out.

Note that by default, Page Objects are positioned vertically, one under the other.

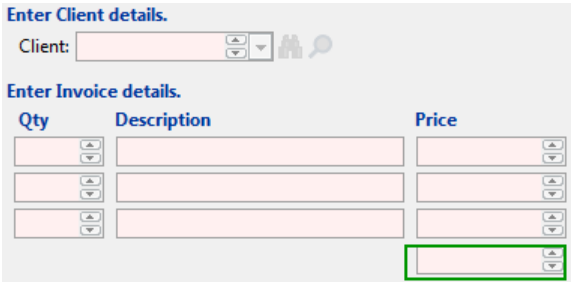
Positioning Page Objects on the Same Line

Checking the 'Keep on the same line as the previous Page Object?' checkbox in a Page Object's properties (show as 'Keep With Previous' in the property grid) has the following affect:



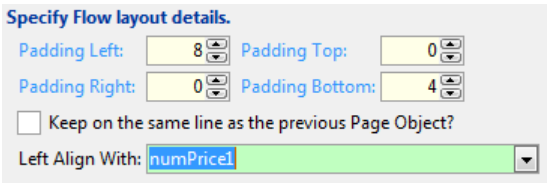
Aligning with an existing Page Object

It is also possible, using the 'Left Align With' property, to align a Page Object with another Page Object that appears before it on the page, e.g.:



In the above example:

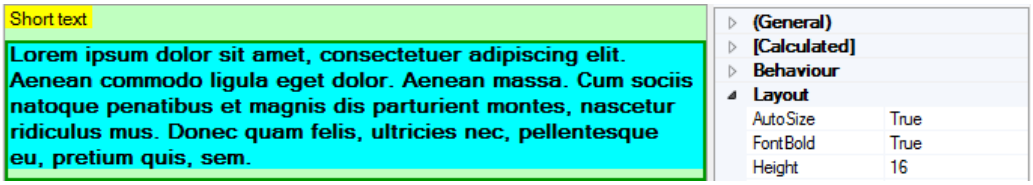
- Each of the 'Description' and 'Price' Page Objects are configured to 'Keep With Previous'.
- The 'Total' NumberBox has been configured to left align with one of the 'Price' NumberBoxes, i.e.:



Auto-Sizing Width

How auto-sizing of a Page Object's width works depends on the type of Page Object. This is detailed more in the Page Objects section but a few examples are:

- Labels will auto-size their width and their height to accommodate their content, e.g.:

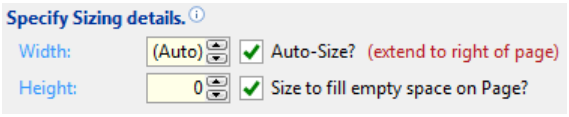


- For TextBoxes, auto-sizing the width will expand the width of the Page Object to extend to the right-hand side of the Page if the 'Max Length' is unspecified. Otherwise, it will size automatically based on the 'Max Length', accounting for the Caption width, e.g.:

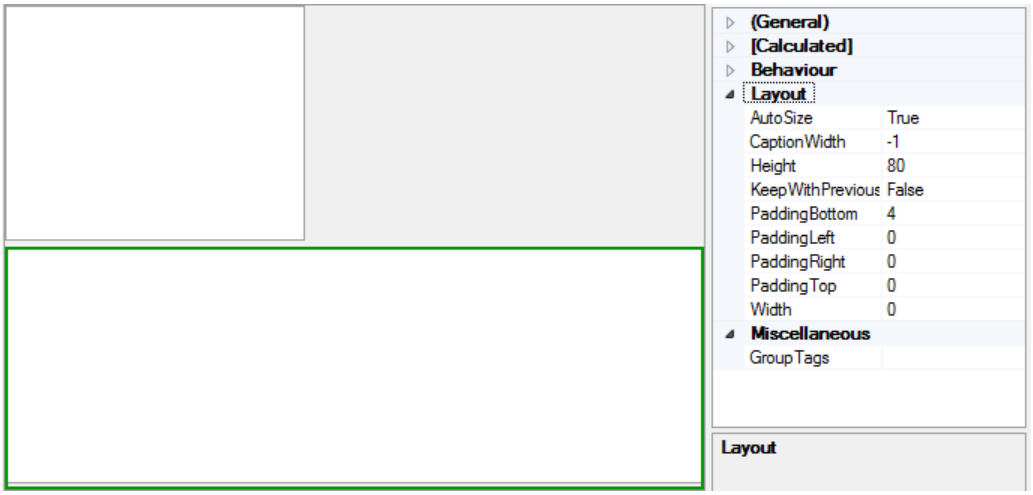
- Grids, HTML Panels and Images will expand their width to extend to the right-hand side of the page but, if a Width is specified, this will be the maximum width they expand to, e.g.:

Auto-Sizing Height to Fill Page

Grids, HTML Panels, Images and multi-line TextBoxes can all be configured to expand their height to fill the page via the 'Size to fill empty space on Page?', e.g.:



If more than one Page Object is set to auto-size height, the empty space on the page will be apportioned equally between the Page Objects, e.g.:



NOTE: If the height of any of the Page Objects is specified, i.e., it is not zero, this will be taken as being the minimum height for the Page Object and it will never be sized smaller than this.

Positioned

Positioned layout mode allows the location and size of all Page Objects to be specified. This allows for the creation of layouts that would not be possible using Flow layout.

Page's that use Positioned layout can utilise the PageResize event to resize or reposition Page Objects.

PageResize Event

This event is called when a Page is first displayed and also when the Page Set form is resized and should generally only be used for Pages using 'Positioned' layout since the `Left` and `Top` properties of a Page Object on a 'Flow' layout page are meaningless, e.g.:

```
Public Sub PageSet_PageResize(sender As Object,
                              e As finPageSetHandlerPageResizeEventArgs) Handles Me.PageResize

  Select Case e.PageId
    Case "POSITIONED"
      imgTest.PageObject.Left = CInt((e.PageWidth - imgTest.PageObject.Width) / 2)
  End Select

End Sub
```

NOTE: The reason why the `Left` property must be accessed from the `.PageObject` property rather than directly from `imgTest` is that we have tried to simplify the number of properties available from the main Page Objects (E.g., `imgTest`) and, since the default layout mode for a Page is 'Flow', you would typically not need to access the `Left` property.

Page Objects

This section details each of the various Page Object types. The main focus will be the Page Object's use and properties for Pages using 'Flow' layout mode.

Various Page Set samples are available which give more practical examples. These are detailed in the Introduction section.

General

Tooltip

Most Page Objects support a Tooltip that is displayed when hovering over the Page Object.

This is typically defined on the Page Object wizard but it can also be updated via Script Code, e.g.:

```
txtFirstName.TooltipText = "Enter the Client's First Name"
```

Field Hint

The following types of Page Object support a Hint that is displayed when the Page Object has no content.

This is useful when displaying multiple Page Objects on a single line with only one caption, e.g.:

You would typically define the Hint on the Page Object wizard but it can also be updated via Script Code, e.g.:

```
txtFirstName.FieldHint = "First Name"
```

Group Tags

A Page Object can define a comma-separated list of Group Tags.

These are useful for performing bulk operations on Page Objects, e.g.:

- You define a set of Page Object relating to a co-borrower in an Account Application Page Set.
- You only want these Page Objects to be visible when the User checks the 'Has Co-Borrower' CheckBox so you assign them all a Group Tags property of 'CoBorrower'.

The following code would achieve the above:

```
Public Sub chkCoBorrower_Change(sender As Object,
                                e As finPageObjectChangeEventArgs) Handles chkCoBorrower.Change

  psh.GroupTagSetVisible("CoBorrower", chkCoBorrower.Value)

End Sub
```

More information on using Group Tags can be found under Advanced Scripting, Group Tags.

Buttons

Many types of Page Objects can define buttons to display. For most types these will be displayed after the text entry portion of the Page Object but for Grids, buttons will be displayed underneath the Grid.

Up to three buttons can be specified in the Page Object wizard, e.g.:

More buttons can be added via Script code (typically in the `Initialise` method). This includes 'Separators' which appear as a vertical line to break up blocks of buttons, e.g.:

```
With txtTextBox1.Buttons
  .Add(isefinPageObjectButtonType.Refresh)
  .Add(isefinPageObjectButtonType.Search)
  .AddSeparator()
  .Add(isefinPageObjectButtonType.Custom, "Copy", "Clipboard_Copy", "Copy to clipboard")
End With
```

Each button has `Enabled` and `Visible` properties allowing their state to be updated, e.g.:

```
txtTextBox1.Buttons.ItemById("Copy").Enabled = False
```

The Page Object's ButtonClick event allows the Page Set Script to respond to a button being clicked, e.g.:

```
Public Sub txtTextBox1_ButtonClick(sender As Object, e As finPageObjectButtonClickEventArgs)
Handles txtTextBox1.ButtonClick

  Select Case e.ButtonType
    Case isefinPageObjectButtonType.Refresh
      ' Do something

    Case isefinPageObjectButtonType.Custom
      Select Case e.ButtonId
        Case "Copy"
          ' Do something
      End Select
  End Select

End Sub
```

NOTE: Non-custom buttons have a `ButtonId` based on their type, e.g., the above could simply have added a `Case "Refresh"` to handle the Refresh button.

Custom buttons can be assigned a shortcut key (typically for buttons that appear after Page Objects such as DBComboBoxes), e.g.:

```
With cboClient.Buttons
  .Add(isefinPageObjectButtonType.Custom, "Load", "Browse_Open",,,,
isefinPageSetShortcutKey.Enter)
  .Add(isefinPageObjectButtonType.Custom, "Search", "Search",,,, isefinPageSetShortcutKey.F6 Or
isefinPageSetShortcutKey.Control)
End With
```

NOTE: The shortcut key is only applicable to Page Objects that can receive input focus, e.g., DBComboBox and not those that can't such as Button Strip.

Non-custom buttons have a built-in Icon, Tooltip and Shortcut key assigned. Custom buttons need to specify their own Icon which should be the resource Id of a finPOWER Connect icon.

The General page of the Page Sets form has an Icon dropdown list that contains all finPOWER Connect icons. The table below lists some of the more common icons used for buttons:

 Select_All	 Select_None	 Select_Up	 Select_Down
 Select_Checked	 Select_Unchecked	 Select_Invert	 Select_Same
 Checkbox_Checked	 Checkbox_Unchecked		
 Clipboard_Copy	 Clipboard_Cut	 Clipboard_Paste	
 Browse_Open	 Form_Open	 Plus	
 Search	 Find		

NOTE: finPOWER Connect icons may appear slightly different on high DPI displays.

Label

Labels are used for headings, information text, hyperlinks etc. The only event supported by labels is the Click event.

Label Styles

Labels have the following pre-defined styles:

- Heading 1
- Heading 2
- Heading 3
- Heading 4
  - This matches the section headings used on built-in finPOWER Connect forms.
- Normal
- Hyperlink
  - Recommended for any labels that are clickable (i.e., handle the Click event) since they are styled like a hyperlink and change the mouse pointer to a hand.
- Warning
- Information

Custom Labels

Custom labels should be used sparingly since they do not conform to any of the built-in finPOWER Connect styles.

Custom labels allow their font and background and foreground colours to be specified; updating these properties from a Page Set Script for non-custom labels with have no effect.

The 'Font Name' should be restricted to a font that you are sure will exist on all Users' PCs. If this font does not exist, the default finPOWER Connect font will be used.

Events

The following events are available for Labels:

- Click
  - The User has clicked the label.

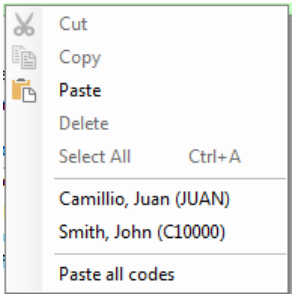
TextBox

TextBoxes should be used for entry of non-numeric, non-date information. The maximum number of characters (Max Length) can be specified.

A TextBox can be configured to be multi-line in which case its height can be specified. It is recommended that this is a multiple of 16.

ContextMenuListObjectType

When right-clicking in a TextBox in certain places within finPOWER Connect, a list of related records will be shown, e.g., right-clicking the Clients range in the Client List report parameters form will show a list of Clients appearing on other open forms, e.g.:



This is achieved by setting the ContextMenuListObjectType to one of the following:

- Account
- AccountApp
- Client
- SecurityStmt
- Workflow

Format

A TextBox can also be given one of the following formats:

- Upper Case
  - The text entered will be converted to upper case.
- Lower Case
  - The text entered will be converted to lower case.
- Proper Case
  - The text entered will be converted to proper case, i.e., the first letter of each word will be capitalised.
  - **NOTE:** Do not use this for name fields since it will not handle exceptions such as 'MacDonald' or 'O'Brien'.
- Phone Number
  - The text entered will be formatted according to phone number formatting rules.

Events

The following events are available for TextBoxes:

- Change
  - The text has been changed by the User.
  - Occurs every time the text is updated.
  - Setting `e.ChangeHandled = True` will update the TextBox's `TextOriginal` property to match the current text.

        - The `TextOriginal` is the value of the text when the TextBox was first focused.
        - **NOTE:** It is recommended that you always set `e.ChangeHandled` to `True` unless using the Enter and Leave events.

- **NOTE:** The Change event is not fired when the Text property is updated in code. If required, you need to specifically handle this after setting the property.

- ButtonClick
    - See Buttons.
- Enter
    - The Page Object has gained input focus.
    - The Page Object's `TextOriginal` property is set to its current `Text` value.
- Leave
    - The Page Object has lost input focus.
    - The Page Object's `TextOriginal` property can be compared against its current `Text` value to see if the content has changed.
        - The `Changed` property will return `True` if these values do not match (but only if `e.ChangeHandled` was not set to True in the Change event).

ComboBox

ComboBoxes are used to show a dropdown list of values. The maximum number of characters (Max Length) can be specified and the value entered into the ComboBox can be limited to the list of dropdown values (Limit to List).

List

The ComboBox's List can be one of the following:

- Information List
  - An Information List.
  - **NOTE:** If the Information list has a Value and an Alt Value, use a DBComboBox.
- SQL Database Query
  - A database query.
- Comma Separated list of items
  - A comma-separated list.
  - If one of the list values contains a comma, it should have quotes around it, e.g.: `Item 1,"Item 2, Second",Item 3`

The list can be updated by the Page Set Script, e.g.:

```
' CSV
cboComboBox1.List = "One,Two,Three"

' Information List
cboComboBox1.List = "LIST.Countries"

' Database Query
cboComboBox1.List = "SQL.Select [AccountId] From [Account]"
```

Events

The following events are available for ComboBoxes:

- Change
  - The text has been changed by the User.
  - Occurs every time the text is updated.
    - If it was updated via selecting an item in the list, `e.ChangedFromPopup` will be `True`.
  - Setting `e.ChangeHandled = True` will update the ComboBox's `TextOriginal` property to match the current text.
    - The `TextOriginal` is the value of the text when the ComboBox was first focused.
    - **NOTE:** It is recommended that you always set `e.ChangeHandled` to `True` unless using the Enter and Leave events.

NOTE: The Change event is not fired when the Text property is updated in code. If required, you need to specifically handle this after setting the property.

- ButtonClick
  - See Buttons.
- Enter
  - The Page Object has gained input focus.
  - The Page Object's `TextOriginal` property is set to its current `Text` value.
- Leave

- The Page Object has lost input focus.
- The Page Object's `TextOriginal` property can be compared against its current `Text` value to see if the content has changed.
  - The `Changed` property will return `True` if these values do not match (but only if `e.ChangeHandled` was not set to True in the Change event).

DBComboBox

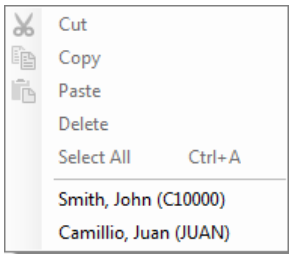
DBComboBoxes (Database ComboBoxes) are used to show a dropdown list of values (E.g., a record Id) and also a description to the right of the text portion.

The maximum number of characters (Max Length) can be specified, as can the width of the text portion (TextBox Width) of the Page Object.

Typically, a DBComboBox will have an auto-sized Width that extends to the right of the page. This accommodates the description to the right of the text portion.

ContextMenuListObjectType

When right-clicking in a DBComboBox in certain places within finPOWER Connect, a list of related records will be shown, e.g., right-clicking the Client field in the Credit Enquiry wizard will show a list of Clients appearing on other open forms, e.g.:



This is achieved by setting the ContextMenuListObjectType to one of the following:

- Account
- AccountApp
- Client
- SecurityStmt
- Workflow

Data Source

The DBComboBox's List (Data Source Type) can be one of the following:

- None
  - The Page Set Script will handle populating the List.
- Standard Range
  - A built-in finPOWER Connect Standard Range.
    - Many Standard Ranges are based on global collections such as Branches. Others such as Accounts and Clients are retrieved directly from the database.
  - A checkbox to only include Active records.
  - The ability to use 'Fast Mode' is given for certain ranges, e.g., Accounts, Clients and Security Statements. This means that the DBComboBox will not display a dropdown button and does not have the overhead or reading a potentially large amount of information from the database.
  - 'Show Find and/ or Search Buttons' allows buttons to be automatically added based on the type of range specified.
- Database Query
  - If a database query is used, an 'Id Field' (the value that appears in the DBComboBox) and, optionally, a 'Desc Field' (Description Field), the description that appears to the right of the text portion should be specified.
- DataView
  - Conceptually the same as 'None' but this just makes it clear that the Script will populate the list from a DataView.

The data source can be updated by the Page Set Script, e.g.:

```
Dim dv As DataView

' Bind to an Information List using special list functionality
If finBL.ResolveListAsDataView("LIST.BankShortNames", dv) Then
  cboComboBox1.RefreshDataView(dv, "Value", "ValueAlt")
End If

' Bind to an Information List using method on finInformationListRO object
' NOTE: This method allows us to control the column names, i.e., change them from Value and
ValueAlt
If finBL.InformationLists("BankShortNames").GetDataView(dv, "Bank", "Description", False) Then
  cboComboBox1.RefreshDataView(dv, "Bank", "Description")
End If

' Bind to a list of Promotions for an Account Type (could also use the
GetAvailablePromotionsDataView method)
If finBL.AccountTypes("VL").Promotions.GetDataView(dv, "") Then
  cboComboBox1.RefreshDataView(dv, "PromotionId", "Description",
"PromotionId,Description,IsAvailable")
End If

' Bind to a Standard Range but override the columns to display
' NOTE: The column names match the properties on the business layer, e.g., finScriptRO properties
cboComboBox1.RefreshStandardRange(isefinStandardRange.Scripts, "", "", False,
"ScriptId,Description,ScriptTypeText")
```

Events

The following events are available for DBComboBoxes:

- Change
  - Occurs every time the text is updated.
    - If it was updated via selecting an item in the list, `e.ChangedFromPopup` will be `True`.
  - Setting `e.ChangeHandled = True` will update the DBComboBoxes's `TextOriginal` property to match the current text.
    - The `TextOriginal` is the value of the text when the DBComboBox was first focused.
    - **NOTE:** It is recommended that you always set `e.ChangeHandled` to `True` unless using the Enter and Leave events.

- **NOTE:** The Change event is not fired when the Text property is updated in code. If required, you need to specifically handle this after setting the property.

- ButtonClick
  - See Buttons.
- Enter
  - The Page Object has gained input focus.
  - The Page Object's `TextOriginal` property is set to its current `Text` value.
- Leave
  - The Page Object has lost input focus.
  - The Page Object's `TextOriginal` property can be compared against its current `Text` value to see if the content has changed.
    - The `Changed` property will return `True` if these values do not match (but only if `e.ChangeHandled` was not set to True in the Change event).

NumberBox

NumberBoxes should be used for entry of numeric information such as currency values.

NumberBoxes can specify Minimum and Maximum values and auto-sizing a NumberBox will size the Page Object to account for these together with the number of decimal places (Decimals) and other settings.

Special Types

A NumberBox can also be one of the following Special Types:

- Currency
  - This will use the number of decimal places based on the database's country settings.
- Percentage
  - This will display a '%' symbol on the end of the number when the Page Object does not have input focus.

Other Properties

A NumberBox also has the following properties that determine how it acts:

- NumberBoxAllowBlank
  - This can be used to indicate that the NumberBox allows a blank value to be stored.
  - Internally, the value is stored as `Double.NaN` but the NumberBox's blank state can be set/ read from a Script using the `ValueIsBlank` property.
  - **NOTE:** A NumberBox's value is never blank unless specifically set from a Script.
- NumberBoxBlankIfZero
  - Indicates whether to show blank in the NumberBox if its value is zero and it does not have input focus.
  - Useful for storing values such as 'Year' where zero is used to indicate that no value has been specified.

Events

The following events are available for NumberBoxes:

- Change
  - Occurs every time the value is changed by either:
    - The spin buttons.
    - The User changing the value and the Page Object losing input focus.

- **NOTE:** The Change event is not fired when the Value property is updated in code. If required, you need to specifically handle this after setting the property.

- ButtonClick
  - See Buttons.
- Enter
  - The Page Object has gained input focus.
  - The Page Object's `ValueOriginal` property is set to its current `Value`.
- Leave
  - The Page Object has lost input focus.
  - The Page Object's `ValueOriginal` property can be compared against its current `Value` to see if the content has changed.

- The `Changed` property will return `True` if these values do not match (but only if `e.ChangeHandled` was not set to True in the Change event).

DateBox

DateBoxes are used for entry of date values.

A Minimum and Maximum value can be specified for the DateBox but generally this will be done via Script code rather than from the Page Object wizard since it is likely to be relative to the current date (E.g., a maximum value which is 30 days in the future).

Special Types

A DateBox can also be one of the following Special Types:

- First Day
  - The date entered must be the first day of a month.
- Last Date
  - The date entered must be the last day of a month.
- Historic
  - The date being accepted is in the past (this is not enforced), e.g., a Date of Birth.
  - If the date is entered with a 2-digit year, this will be assumed to be in the past, e.g., a year of '89' will be converted to '1989'.

Events

The following events are available for DateBoxes:

- Change
  - Occurs every time the value is changed by either:
    - The spin buttons.
    - The dropdown calendar.
    - The User changing the value and the Page Object losing input focus.

- **NOTE:** The Change event is not fired when the Value property is updated in code. If required, you need to specifically handle this after setting the property.

- ButtonClick
  - See Buttons.
- Enter
  - The Page Object has gained input focus.
  - The Page Object's `ValueOriginal` property is set to its current `Value`.
- Leave
  - The Page Object has lost input focus.
  - The Page Object's `ValueOriginal` property can be compared against its current `Value` to see if the content has changed.
    - The `Changed` property will return `True` if these values do not match (but only if `e.ChangeHandled` was not set to True in the Change event).

Date Cycle ComboBox

Date Cycle ComboBoxes are used for entry of terms and frequencies, e.g., '12 Years' or 'Fortnightly'.

By default, a Date Cycle ComboBox will assume that a term is being entered, e.g., '12 years'. Checking the 'Frequency?' checkbox will configure the Page Object to accept a frequency, e.g., 'Fortnightly'.

Typically, a Date Cycle ComboBox will have an auto-sized Width that extends to the right of the page. This accommodates the description to the right of the text portion.

List

A comma-separated list of terms or frequencies can be entered for the Date Cycle ComboBox and the Page Object can restrict data-entry to this list (Limit to List).

Events

The following events are available for Date Cycle ComboBoxes:

- Change
  - Occurs every time the value is changed by either:
    - Selecting a value from the dropdown list.
    - The User changing the value and the Page Object losing input focus.

- **NOTE:** The Change event is not fired when the Text property is updated in code. If required, you need to specifically handle this after setting the property.

- Enter
  - The Page Object has gained input focus.
  - The Page Object's `TextOriginal` property is set to its current `Text` value.
- Leave
  - The Page Object has lost input focus.
  - The Page Object's `TextOriginal` property can be compared against its current `Text` value to see if the content has changed.
    - The `Changed` property will return `True` if these values do not match (but only if `e.ChangeHandled` was not set to True in the Change event).

DateTimeZone

DateTimeZones are used for entry of date and time values and, if the finPOWER Connect database is configured to use Time Zones, a Time Zone can also be entered.

Since this is a composite Page Object, it exposes both `Value` and `TimeZoneId` properties.

Events

The following events are available for DateTimeZones:

- Change
  - Occurs every time the value is changed by either:
    - The spin buttons.
    - The dropdown calendar.
    - The Time Zone dropdown.
    - The User changing the value and the Page Object losing input focus.
  - **NOTE:** Since this a composite Page Object (i.e., it has both a Value and `TimeZoneId` property), there is no `ValueOriginal` or Changed properties.

- **NOTE:** The Change event is not fired when the Value property is updated in code. If required, you need to specifically handle this after setting the property.

CheckBox

CheckBoxes are used for entry of Boolean values.

Option Button Style

A CheckBox can be styled as an 'Option button', i.e., a circle rather than a square.

This is typically used for a group of CheckBoxes of which only one can be checked at one time.

The following example achieves this for three CheckBoxes:

```
Public Sub chkCheckBox_Change(sender As Object,
                              e As finPageObjectChangeEventArgs) Handles chkCheckBox1.Change,
chkCheckBox2.Change, chkCheckBox3.Change

  chkCheckBox1.Value = sender Is chkCheckBox1
  chkCheckBox2.Value = sender Is chkCheckBox2
  chkCheckBox3.Value = sender Is chkCheckBox3

End Sub
```

NOTE: The same Change event handler is used for all three CheckBoxes.

Events

The following events are available for CheckBoxes:

- Change
  - Occurs every time the value is changed by the User.

- **NOTE:** The Change event is not fired when the Value property is updated in code. If required, you need to specifically handle this after setting the property.

Button

Buttons can be auto-sized based upon their caption.

Events

The following events are available for Buttons:

- Click
  - Occurs when the button is clicked.

HTML Editor

An HTML Panel is used to enter either plain text or HTML, for sending an HTML Email.

When auto-sizing an HTML Editor, you would typically set the Width to zero which sizes the HTML Editor to fit the Page width. Specifying a non-zero Width will size the Page Object to fit the Page width but restrict its maximum Width to the value specified.

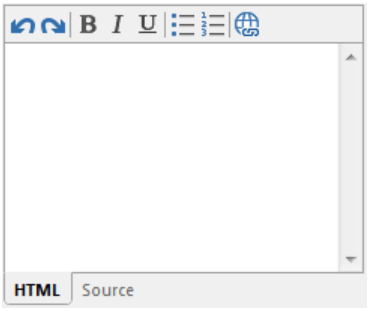
If shown at the bottom of the Page, the 'Size to fill empty space on Page' option can be checked to extend the Page Object to fill Page vertically.

HTML can be entered directly in the editor on the HTML tag, including pasting from a source such as Microsoft Word (although the HTML pasted will be very verbose).

The Source tab can be used if more control over the HTML is required.

Formatting Toolbar

By default, an HTML Editor will display a formatting toolbar:



This toolbar is not configurable and can be hidden by unchecking the 'Show Formatting Toolbar' option on the HTML Editor page of the Page Object wizard.

Hyperlinks

By default, even when the Page Object is read-only, clicking a hyperlink will do nothing.

However, version 2.03.03 introduced a new setting to 'Allow hyperlinks to be followed when read-only'. If this is set then any HTTP or HTTPS hyperlinks in the HTML will be opened using the default associated application (generally a Web browsers such as Internet Explorer or Chrome).

Events

The following events are available for HTML Editors:

- Change
  - The text has been changed by the User.
  - Occurs only when the Page Object loses focus.
  - Setting `e.ChangeHandled = True` will update the HTML Editor's `TextOriginal` property to match the current text.
    - The `TextOriginal` is the value of the text when the HTML Editor was first focused.
    - **NOTE:** It is recommended that you always set `e.ChangeHandled` to `True` unless using the Enter and Leave events.

- **NOTE:** The Change event is not fired when the Text property is updated in code. If required, you need to specifically handle this after setting the property.

- Enter
  - The Page Object has gained input focus.
  - The Page Object's `TextOriginal` property is set to its current `Text` value.

- Leave
  - The Page Object has lost input focus.
  - The Page Object's `TextOriginal` property can be compared against its current `Text` value to see if the content has changed.
    - The `Changed` property will return `True` if these values do not match (but only if `e.ChangeHandled` was not set to True in the Change event).

NOTE: Behind the scenes, this Page Object uses Internet Explorer's editing capabilities.

The HTML produced is a little antiquated (upper case tags are generated) and is outside of the control of Intersoft Systems.

HTML Panel

An HTML Panel is used to display HTML, e.g., a Summary Page.

When auto-sizing an HTML Panel, you would typically set the Width to zero which sizes the HTML Panel to fit the Page width. Specifying a non-zero Width will size the Page Object to fit the Page width but restrict its maximum Width to the value specified.

HTML Panels are often shown at the bottom of the Page (E.g., under a Grid) and are required to fill any remaining space on the Page. In this case the 'Size to fill empty space on Page' option should be checked.

An HTML Panel can also be given a 'Disabled Caption' which will display in place of the HTML if the Page Object is disabled.

Printing

As per any HTML page within finPOWER Connect, HTML Panel type Page Objects can be printed by right-clicking on the page and selecting the Print option.

HTML Panels can also be printed from Script code, e.g.:

```
Public Sub cmdPrint_Click(sender As Object,
                          e As finPageObjectClickEventArgs) Handles cmdPrint.Click

  htmlPanel.Print()

End Sub
```

Hiding the Border

The border of an HTML Panel can be hidden via the Page Object wizard.

This is useful for creating 'rich' content on the Page Set that looks like it is part of the Page Set rather than HTML.

The following example shows how to populate the content of an HTML Panel so that the HTML blends in with the Page Set, including displaying a built-in finPOWER Connect icon at 32x32 pixels and setting the background colour to the 'Window' system colour (this changes based on the selected finPOWER Connect theme):

```
Private Sub htmlPanel1_Refresh()

  Dim sb As StringBuilder

  ' Initialise
  sb = New StringBuilder()

  ' Create HTML
  ' NOTE: CSS can be used to set background to Windows system colour
  sb.AppendLine("<!DOCTYPE html>")
  sb.AppendLine("<html style='overflow:hidden'>")
  sb.AppendLine("<body style='padding:0; margin:0; background-color:{{SystemColour|Window}}>")

  sb.Append("<table border='0' cellpadding='0' cellspacing='0'>")
  sb.Append("<tr>")
  sb.Append("<td style='padding-right:8px'>")
  sb.Append("<img src='" &
finBL.Utilities.GetIconAsPngBase64String("Account", 32, "Warning") & "' width='32' height='32'/>")
  sb.Append("</td>")
  sb.Append("<td>")
  sb.Append("<div style='font:16pt Segoe UI; font-weight:bold; color:#3399FF'>My Heading</div>")
  sb.Append("<div style='font:9pt Segoe UI'>Other text below the heading</div>")
  sb.Append("</td>")
  sb.Append("</tr>")
  sb.Append("</table>")

  sb.AppendLine("</body>")
  sb.AppendLine("</html>")

  ' Update HTML
  htmlPanel1.Text = sb.ToString()

End Sub
```

Events

The following events are available for HTML Panels:

- CustomHyperlinkClick
  - Occurs when a hyperlink in the HTML is clicked.
    - A custom hyperlink should begin either '`custom://`' or '`script://`'.
    - The event handler is passed an Application Shortcut from which the Script can handle the action and also access any parameters.
    - The following code sample sets an HTML Panel's text and then displays information about the application shortcut when it is clicked:

```
Public Overrides Function Initialise() As Boolean

  ' Assume Success
  Initialise = True

  ' Initialise
  mReports = DirectCast(psh.Reports, ISfinReports)
  mUI = DirectCast(psh.UserInterface, ISUserInterfaceBL)

  htmlPanel1.Text = "<a href='custom://mytest?name=paul&age=25'>test</a>"

End Function

Public Sub htmlPanel1_CustomHyperlinkClick(sender As Object,
                                           e As
finPageObjectCustomHyperlinkClickEventArgs) Handles htmlPanel1.CustomHyperlinkClick

  mUI.MsgBox(e.ApplicationShortcut.Action & vbNewLine &
e.ApplicationShortcut.Parameters.GetString("Name"))

End Sub
```

Grid

A Grid is used to display tabular information.

When auto-sizing a Grid, you would typically set the Width to zero which sizes the Grid to fit the Page width. Specifying a non-zero Width will size the Page Object to fit the Page width but restrict its maximum Width to the value specified.

Grids are often shown at the bottom of the Page or are required to fill any remaining space on the Page. In this case the 'Size to fill empty space on Page' option should be checked.

Columns

Grid columns can only be defined in the Page Set Script. Typically this would be done in the `Initialise` method, e.g.:

```
' Grids
With gridTest
  With .Columns
    .AddDrilldown("Drilldown")
    .AddIcon("Icon", "Icon")
    .AddBoolean("Selected", "Selected", 20, False)
    .AddString("AccountId", "Code", 80, True)
    .AddString("Name", "Name", 100, True)
  End With
End With
```

Once a column has been added, many properties such as `Key` and `ReadOnly` cannot be changed. The following properties can however be updated:

- Visible
  - Can be updated to show or hide the column dynamically.
- Caption
  - Can be updated to change the column's caption.

Grids support the following column types:

- Boolean
- Currency
- Date
- DateTime
  - Always read-only.
- Double
- Drilldown
  - Always read-only.
  - The Key should always start with the word 'Drilldown'. If it does not, it will be prefixed with 'Drilldown_'.
- Icon
  - Always read-only.
  - Preferably, name any icon columns with a name ending in 'Icon', e.g., StatusIcon. This will prevent the column from being included in any grid exports.
  - Setting the grid cell's value (see Data Binding) to a valid icon resource Id will display that icon in the cell.
    - **NOTE:** A list of icons can be viewed from the 'Icon' dropdown on the General page of either the Page Sets or Workflow Types forms.
    - Special, dynamically coloured icons can be created to display flags, bullets and colour blocks. These include the HTML colour in square brackets (e.g., Red or #FF0000) after the resource Id, e.g.:
      - `Flag[Red]`

        - `Bullet[#FF0000]`
        - `Colour[#FF0000]`
    - If the value contains a comma, the text after the comma will be assumed to be the name of an overlay icon, e.g.:
        - `Account,Add`
        - `Client,Search`
- Integer
- Percent
    - Always read-only.
- String
    - Normal
    - ListCsv
        - Column can specify a simple CSV list and also a `LimitToList` parameter.

NOTE: Column alignment is automatic based on the column type.

Groupings

Grid columns can be grouped using the Grid's `GroupByColumns` property as shown below:

```
Public Overrides Function Initialise() As Boolean

  ' Standard code omitted for clarity

  ' Grids
  With gridTest
    With .Columns
      .AddString("ClientId", "Code", 80, True)
      .AddString("Name", "Name", 200, True)
      .AddString("AccountRoleId", "Role", 80, True)
    End With

    .GroupByColumns = "AccountRoleId"
  End With

End Function
```

NOTE: You would typically only group by a single column but `GroupByColumns` can accept a comma-separated list of Column keys.

Data Binding

Grid data is displayed based on a 'Virtual Data Binding' model.

The grid performs a `VirtualDataBind` to a data source which can be either a DataView or collection (DataSet and DataTable can be used but all this does is bind to the DefaultView). The grid's `InitialiseRow` event is then called for each item in the collection. It is this event that the Page Set Script must use to populate the grid cells, e.g.:

```
Private mAccount As finAccount

Public Overrides Function Initialise() As Boolean

  ' Standard code omitted for clarity

  ' Load Account
  mAccount = finBL.CreateAccount()
  Initialise = mAccount.Load("L10000")

  ' Grids
  With gridTest
    .ShowRowNumbering = True
```

```
    With .Columns
      .AddDrilldown("Drilldown")
      .AddString("ClientId", "Code", 80, True)
      .AddString("Name", "Name", 200, True)
      .AddString("AccountRoleId", "Role", 80, True)
      .AddString("Notes", "Notes", 100)
    End With
  End With

  ' Bind to grid
  gridTest.VirtualDataBind(mAccount.Clients)

End Function

Public Sub gridTest_InitialiseRow(sender As Object,
                                  e As finPageObjectInitialiseRowEventArgs) Handles
gridTest.InitialiseRow

  If e.ListIndex >= 0 AndAlso e.ListIndex < mAccount.Clients.Count Then
    ' Update Fields
    With mAccount.Clients(e.ListIndex)
        e.Row.Cells("ClientId").Value = .ClientId
        e.Row.Cells("Name").Value = .ClientName
        e.Row.Cells("AccountRoleId").Value = .AccountRoleId
        e.Row.Cells("Notes").Value = .Notes
    End With
  End If

End Sub
```

The `InitialiseRow` event can also be used to style a row or a cell, e.g., to update the background or foreground colours, add a tooltip or force a cell to read-only.

It can also be used to hide a row:

```
Public Sub gridTest_InitialiseRow(sender As Object,
                                  e As finPageObjectInitialiseRowEventArgs) Handles
gridTest.InitialiseRow

  If e.ListIndex >= 0 AndAlso e.ListIndex < mAccount.Clients.Count Then
    With mAccount.Clients(e.ListIndex)
      ' Update Fields
      e.Row.Cells("ClientId").Value = .ClientId
      e.Row.Cells("Name").Value = .ClientName
      e.Row.Cells("AccountRoleId").Value = .AccountRoleId
      e.Row.Cells("Notes").Value = .Notes

      ' Style
      If .IsOwner Then e.Row.ColourBackground = "yellow"
      If .RoleJoint Then e.Row.Cells("AccountRoleId").ColourForeground = "#0000ff"

      ' Hide Row
      If .Active Then e.Row.Visible = False

      ' Tooltip
      e.Row.Cells("ClientId").TooltipText = "Credit Rating: " & .Client.CreditRating

      ' Read-Only
      If .IsOwner Then
        e.Row.Cells("Notes").ReadOnly = iseDefaultableBoolean.True
      End If
    End With
  End If

End Sub
```

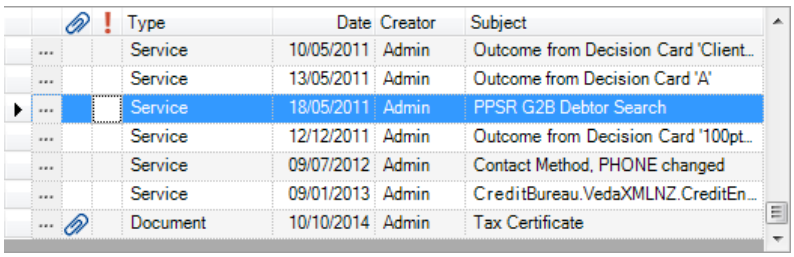
You can re-bind the data in the grid using the `VirtualDataRefresh` method, e.g., if the underlying data source has changed:

```
gridTest.VirtualDataRefresh(-2, "", True)
```

`VirtualDataRefresh` takes the following parameters:

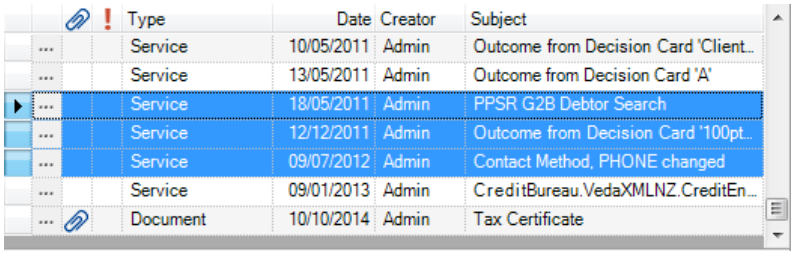
- `newActiveRowIndex`

- The index of the row to activate or:
  - -1 to not activate any rows.
  - -2 to retain the existing active row index.
- **NOTE:** The active row is indicates by an arrow in the row selectors, e.g.:



	Type	Date	Creator	Subject
...	Service	10/05/2011	Admin	Outcome from Decision Card 'Client...
...	Service	13/05/2011	Admin	Outcome from Decision Card 'A'
▶ ...	Service	18/05/2011	Admin	PPSR G2B Debtor Search
...	Service	12/12/2011	Admin	Outcome from Decision Card '100pt...
...	Service	09/07/2012	Admin	Contact Method, PHONE changed
...	Service	09/01/2013	Admin	CreditBureau.VedaXMLNZ.CreditEn...
...	Document	10/10/2014	Admin	Tax Certificate

- `newActiveColumnKey`
  - The key of the column to select or:
    - "*" to retain the current selected column.
    - "" for unspecified, i.e., the first column.
- `retainSelectedRows`
  - Specifying `True` will retain the selected rows, i.e., the rows with their column selector selected:



	Type	Date	Creator	Subject
...	Service	10/05/2011	Admin	Outcome from Decision Card 'Client...
...	Service	13/05/2011	Admin	Outcome from Decision Card 'A'
▶ ...	Service	18/05/2011	Admin	PPSR G2B Debtor Search
...	Service	12/12/2011	Admin	Outcome from Decision Card '100pt...
...	Service	09/07/2012	Admin	Contact Method, PHONE changed
...	Service	09/01/2013	Admin	CreditBureau.VedaXMLNZ.CreditEn...
...	Document	10/10/2014	Admin	Tax Certificate

Updating Cell Values

The `BeforeCellUpdate` event is used to update the underlying data source when the User updates a cell value.

This event is only called for non-read-only columns, e.g.:

```
Public Sub gridTest_BeforeCellUpdate(sender As Object,
                                     e As finPageObjectBeforeCellUpdateEventArgs) Handles
gridTest.BeforeCellUpdate

  If e.ListIndex >= 0 AndAlso e.ListIndex < mAccount.Clients.Count Then
    Select Case e.ColumnKey
      Case "Notes"
        mAccount.Clients(e.ListIndex).Notes = CStr(e.NewValue)
    End Select
  End If

End Sub
```

When updating the underlying data source, `e.NewValue` must be cast to the correct data type, e.g.:

```
Public Sub gridData_BeforeCellUpdate(sender As Object,
                                     e As finPageObjectBeforeCellUpdateEventArgs) Handles
gridData.BeforeCellUpdate

  With DirectCast(mItems(e.ListIndex + 1), Item)
    Select Case e.ColumnKey
      Case "BooleanValue"
        .BooleanValue = CBool(e.NewValue)
      Case "CurrencyValue"
        .CurrencyValue = finBL.Runtime.NumberUtilities.ConvertToCurrency(e.NewValue)
      Case "DateValue"
        .DateValue = finBL.Runtime.DateUtilities.ConvertToDate(e.NewValue)
      Case "DoubleValue"
        .DoubleValue = finBL.Runtime.NumberUtilities.ConvertToDouble(e.NewValue, 4)
      Case "IntegerValue"
        .IntegerValue = finBL.Runtime.NumberUtilities.ConvertToInteger(e.NewValue)
      Case "StringValue"
        .StringValue = CStr(e.NewValue)
    End Select
  End With

End Sub
```

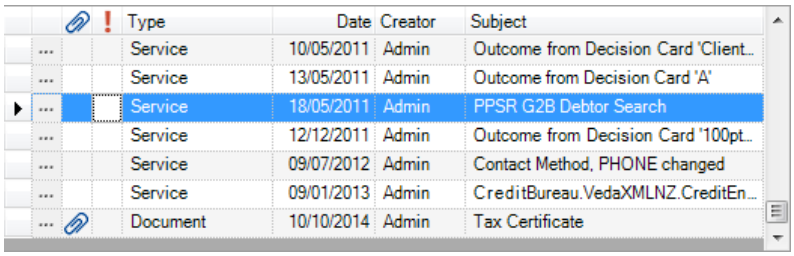
Row Selection

The grid's current (active) row can be determined and set via the grid's `ActiveDataRowIndex` property. This will be -1 if there is no active row in the grid. The `AfterRowActivate` event is fired whenever the active row changes.

The grid's selected rows are distinct from its active row. Rows are selected by clicking the row selectors or by calling one of the following grid methods:

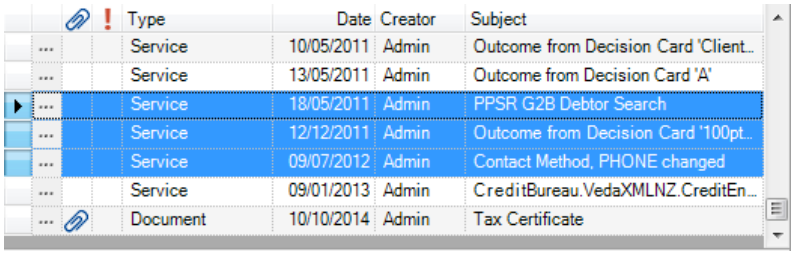
- `SelectedRowsClear()`
  - Clear row selection.
- `SelectRow(index, selected)`
  - Select or unselect the row with the specified index.
- `SelectAllRows()`
  - Select all rows.
- `SelectRowsUp()`
  - Select all rows above and including the active row.
- `SelectRowsDown()`
  - Select all rows below and including the active row.

A grid's active row is indicated by a right-facing arrow:



	Type	Date	Creator	Subject
...	Service	10/05/2011	Admin	Outcome from Decision Card 'Client...
...	Service	13/05/2011	Admin	Outcome from Decision Card 'A'
▶ ...	Service	18/05/2011	Admin	PPSR G2B Debtor Search
...	Service	12/12/2011	Admin	Outcome from Decision Card '100pt...
...	Service	09/07/2012	Admin	Contact Method, PHONE changed
...	Service	09/01/2013	Admin	CreditBureau.VedaXMLNZ.CreditEn...
...	Document	10/10/2014	Admin	Tax Certificate

Whereas, selected rows have their row selectors (the left-most column) highlighted:



	Type	Date	Creator	Subject
...	Service	10/05/2011	Admin	Outcome from Decision Card 'Client...
...	Service	13/05/2011	Admin	Outcome from Decision Card 'A'
▶ ...	Service	18/05/2011	Admin	PPSR G2B Debtor Search
...	Service	12/12/2011	Admin	Outcome from Decision Card '100pt...
...	Service	09/07/2012	Admin	Contact Method, PHONE changed
...	Service	09/01/2013	Admin	CreditBureau.VedaXMLNZ.CreditEn...
...	Document	10/10/2014	Admin	Tax Certificate

The grid's `GetSelectedRows()` method returns an Integer array containing the indexes of the selected rows. The following example updates a label to show these indexes:

```
Public Sub gridTest_AfterSelectedRowsChanged(sender As Object,
                                             e As finPageObjectAfterSelectedRowsChangedEventArgs)
Handles gridTest.AfterSelectedRowsChanged

  Dim i As Integer
  Dim strTemp As String

  For Each i In gridTest.GetSelectedRowIndexes()
    If Len(strTemp) <> 0 Then strTemp &= ","
    strTemp &= CStr(i)
  Next

  lblSelectedRows.Text = strTemp

End Sub
```

Printing

As per any grid within finPOWER Connect, Grid type Page Objects can be printed by right-clicking on the Grid and selecting the Print option.

Grids can also be printed from Script code, e.g.:

```
Public Sub cmdPrint_Click(sender As Object,
                          e As finPageObjectClickEventArgs) Handles cmdPrint.Click

  gridTest.Print()

End Sub
```

Saving and Loading Grid Layout

The grid's layout, e.g., its column widths can be saved and loaded from the Page Set Script.

The following methods handle the grid's layout:

- `LayoutSave()`
  - Save the grid's layout.
    - **NOTE:** As per normal form layouts, this is saved on a per-User basis to the finPOWER Connect's Registry database table.
- `LayoutLoad()`
  - Load the grid's layout.

        - **NOTE:** If no layout has previously been saved, this will simply reset the grid's layout to its initial state.
- `LayoutReset()`
    - Reset the grid's layout to its initial state.
- `LayoutClearSaved()`
    - Clear the grid's previously saved layout and reset the grid's layout to its initial state.

NOTE: Typically, you would use a button below the grid to save the grid's layout and then restore the saved layout using the `LayoutLoad()` method in the Page Set Script's `Initialise()` method.

Other Properties

The following grid properties can also be set to control the grid's appearance. These should be set in the `Initialise` method when the grid is first configured:

- `ShowRowNumbering = True`
    - Show the row number in the row selector.
- `ShowSelectionOpaque = True`
    - Causes the selected and active row colouring to become opaque so that the grid row's colour can be seen regardless of whether a row is selected or not. This is used on certain grids within finPOWER Connect such as Logs and Task Manager grids.

Events

The following events are available for Grids:

- AfterRowActivate
    - Occurs after the current (active) grid row has changed.
    - Use this event for functionality such as updating an HTML Summary Page for the active grid row.
- AfterSelectedRowsChanged
    - Occurs after the row selection has changed.
    - Use this event for functionality such as updating grid button states, e.g., a 'Delete' button that acts on all selected rows and should therefore be disabled if no rows are selected.

      WARNING: Do not use this event for updating information based on the currently active grid row. Use the AfterRowActivate event instead.
- BeforeCellUpdate
    - Occurs when a cell's value has been updated by the User.
- InitialiseRow
    - Occurs for each row in the grid and allows the grid's cell values to be set.
- RowDrilldown
    - Occurs when a drilldown button is clicked.
- ButtonClick
    - See Buttons.

FAQ

- **How do I make a cell read-only?**

- Use the `InitialiseRow` event to change the read-only state of the cell, e.g.:

```
e.Row.Cells("ClientId").ReadOnly = iseDefaultableBoolean.True
```

NOTE: You cannot make a cell in a read-only column not read-only.

- **How do I set a cell's icon.**
  - Use the `InitialiseRow` event to change the icon-type column's cell value to a valid icon resource Id, e.g.:

```
e.Row.Cells("Icon").Value = "Account"          ' Account icon
e.Row.Cells("Icon").Value = "Flag[Red]"        ' Red flag
e.Row.Cells("Icon").Value = "Colour[#00FF00]"  ' Green colour block
```

- **How do I define what happens when I click a drilldown button?**
  - Use the `RowDrilldown` event and act based on the column's key, e.g.:

```
Select Case e.ColumnKey
  Case "Drilldown"
    ' Show Clients form
    finBL.ExecuteApplicationShortcutUrl("FormShow?form=Clients&id=" &
finBL.Runtime.HtmlUtilities.UrlEncode(mItems(e.ListIndex).ClientId))

  Case "Drilldown_Account"
    ' Show Accounts form
    finBL.ExecuteApplicationShortcutUrl("FormShow?form=Accounts&id=" &
finBL.Runtime.HtmlUtilities.UrlEncode(mItems(e.ListIndex).AccountId))
End Select
```

NOTE: The Id of all drilldown columns is prefixed by 'Drilldown'

- **How do I show row numbering on a grid?**
  - Set the grid's `ShowRowNumbering` property, e.g.:

```
gridTest.ShowRowNumbering = True
```

- **How do I clear a grid?**
  - Typically to clear a grid, you would clear the items from the collection or rows from the data table that the grid is bound to and then call the `VirtualDataRefresh()` method, e.g.:

```
CollectionForGrid.Clear()
gridTest.VirtualDataRefresh()
```

- **How do I show or hide a grid column?**
  - Use the `Visible` property of a grid's column to hide or show it, e.g.:

```
gridAccounts.Columns("Description").Visible = False
```

- **How do I clear all selected rows and the active row?**
  - To clear all selected rows, use the grid's `SelectedRowsClear()` method.
  - To clear the active row, set the grid's `ActiveDataRowIndex` property to -1.

```
gridAccounts.SelectedRowsClear()
gridAccounts.ActiveDataRowIndex = -1
```

Image

An Image type Page Object is used to display an image such as a company logo or a built-in finPOWER Connect icon.

When specifying an Image's file name, you can use either a local or network filename or a URL.

The image should be either a GIF, JPEG, ICO or PNG type image.

An image can be embedded into the Page Set. Once embedded, the file name is no longer required.

A preview of the image is displayed in the Page Object wizard and from here, the Page Object's size can be updated via a hyperlink to match the image size:

Image Size Modes

By default, the Image will be displayed at its actual size meaning that, depending on the Page Object size, it may appear clipped.

The following Size Modes can be specified:

- Normal
  - Default. The image will be clipped if the Page Object is too small to fit it. If the Page Object is larger than the image, white-space will appear to the right and below the image.
- Center
  - The image will be centered both horizontally and vertically within the Page Object.
- Stretch
  - The image will be stretched to fit the Page Object. This may lead to the image becoming distorted if the Page Object's aspect ratio does not match the image's aspect ratio.
- Zoom

- The image will be zoomed to fit the Page Object. The image will not be distorted since its aspect ratio will be retained. This may lead to white space to the right and below the image depending on the Page Object's aspect ratio.

Updating the Image

Script code can update the image by specifying a different file name or URL, e.g.:

```
Public Sub cmdTest_Click(sender As Object,
                         e As finPageObjectClickEventArgs) Handles cmdTest.Click

  imgTest.ImageFileName = "http://www.intersoft.co.nz/images/intersoft.gif"
  imgTest.ImageSizeMode = isefinPageObjectImageSizeMode.Zoom

End Sub
```

Or, by specifying a built-in icon and, optionally, an overlay icon, e.g.:

```
Public Sub cmdTest_Click(sender As Object,
                         e As finPageObjectClickEventArgs) Handles cmdTest.Click

  imgTest.ImageFileName = "Account,Search "

End Sub
```

NOTE: If the Image File Name does not contain a dot or backslash, it is assumed to be a built-in Icon.

Events

The following events are available for Images:

- Click
  - Occurs when the Image is clicked.

Button Strip

An Button Strip type Page Object is used to display a toolbar or a strip of buttons under another Page Set such as an HTML Panel (Grids have their own, built-in Button Strip).

Auto-sizing of a Button Strip is as per Grids and HTML Panels, i.e., they will size to fit the Page Width if their Width is set to zero or, if the Width is non-zero, they will size to fit the Page Width up to a maximum of the specified Width.

Buttons

The Buttons collection is the same as other Page Objects and is detailed in the Buttons section.

Border Style

A Button Strip can have one of the following border styles:

- None
  - No border.
- Bottom Only
  - A border will appear beneath the Button Strip only.
    - This is useful if the Button Strip is positioned at the top of a Page, e.g., if will look more like a toolbar.
- Bottom, Left and Right
  - A border will appear at the bottom and left and right of the Button Strip.
  - This is useful if the Button Strip is positioned beneath another Page Object and should appear as being 'joined' to the Page Object above (E.g., an HTML Panel of a multi-line TextBox).
- Top Only
  - A border will appear at the top of the Button Strip only.
  - This is useful if the Button Strip is positioned at the bottom of a Page but should not appear as being 'joined' to the Page Object above (E.g., an HTML Panel of a multi-line TextBox).

Events

The following events are available for Button Strips:

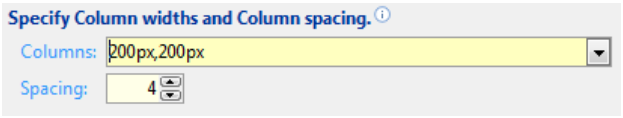
- ButtonClick
  - See Buttons.

Columns Start

A Columns Start type Page Object is used to start a multi-column section within a 'Flow' layout Page.

For an example of a multi-column layout, see Using Column Page Objects.

The number of columns and, optionally, the widths of the columns can be specified in the Page Object wizard, e.g.:



The 'Columns' are specified as a comma-separated list with an entry for each column. Column widths can be defined as:

- Wildcard '*'
  - Apportions columns equally, e.g., ***,*,***
    - Creates 3 columns of equal width.
  - Takes up the remaining space, e.g., **300px,***
    - Create 2 columns, the first being 300 pixels wide and the second taking up the remaining Page width.
- Pixel values
  - Sets a column to the specified number of pixels, e.g., **300px,200px,200px**
    - Creates 3 columns, the first being 300 pixels wide and the next two being 200 pixels wide.
- Percentages
  - Gives each column a percentage of the Page width, e.g., **60%,40%**
    - Creates 2 columns, the first being 60% of the Page width and the second, 40%.
- A minimum column width can also be specified using square brackes, e.g.:
  - ***[200px],***
    - Creates 2 columns of equal width under normal circumstances but ensures that the first column is never less than 200 pixels wide.
      - If the Page is 500 pixels wide, each column would be (excluding spacing), 250 pixels.
      - If the Page is 350 pixels wide, the first column would be 200 pixels wide and the second 150 pixels (excluding spacing).

WARNING: Do not attempt to nest multi-column sections within each other; this will lead to unpredictable results and is not supported.

Column Break

A Column Break type Page Object is used to denote the end of a column in a multi-column section within a 'Flow' layout Page.

For an example of a multi-column layout, see Using Column Page Objects.

The number of column breaks must match the number of columns defined in the Columns Start Page Object.

Account Payment Details

An Account Payment Details type Page Object is used to display or enter Payment Details for an Account.

This Page Object consists of multiple controls which cannot be accessed independently.

Currently, the caption width cannot be set for this type of Page Object, they are set at 80 pixels. Also, internally, all controls are indented by 8 pixels.

Showing Account Payment Details

The following code example shows how to populate this Page Object from an Account's Payment Details. The page contains a DBComboBox named cboAccounts and the Payment Details are updated whenever this is changed:

```
' Objects
Private mAccount As finAccount

Public Overrides Function Initialise() As Boolean

  ' Assume Success
  Initialise = True

  ' Initialise
  mReports = DirectCast(psh.Reports, ISfinReports)
  mUI = DirectCast(psh.UserInterface, ISUserInterfaceBL)

  ' Create Objects
  mAccount = finBL.CreateAccount()

  ' Controls
  With pdaPaymentDetails
    .PaymentFlowDirection = isefinPaymentMethodPaymentFlowDirection.Incoming
    .Account = mAccount
  End With

End Function

Public Sub cboAccount_Change(sender As Object,
                             e As finPageObjectChangeEventArgs) Handles cboAccount.Change

  ' Handled
  If e IsNot Nothing Then e.ChangeHandled = True

  ' Load Account
  If Len(cboAccount.Text) = 0 Then
    mAccount.Clear()
  Else
    If Not mAccount.Load(cboAccount.Text) Then
      mUI.ErrorMessageShow()
    End If
  End If

  ' Upate Payment Details
  pdaPaymentDetails.BankingDetails =
mAccount.BankingDetails(isefinAccountBankingDetailsType.Incoming).ToBankingDetailsForDisplay(False
)

End Sub
```

Reading Account Payment Details

The values displayed in this Page Object can be accessed via the `BankingDetails` property of the Page Object.

Events

The following events are available for Account Payment Details:

- Change
  - Occurs when any of the controls within this Page Object are changed.

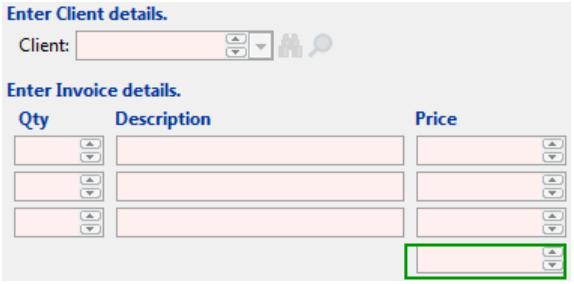
Advanced Layouts

Multi-Column Layouts

Multi-column layouts can be useful, e.g., for presenting a table of information such as the entry of multiple cheque details, or, for preventing a Page from becoming too long, e.g., by presenting Main and Joint borrower details next to each other in an Account Application.

Using Page Objects Kept on the Same Line

Keeping Page Objects on the same line and ensuring their widths are the same allows a table-like effect to be achieved, e.g.:



In this example:

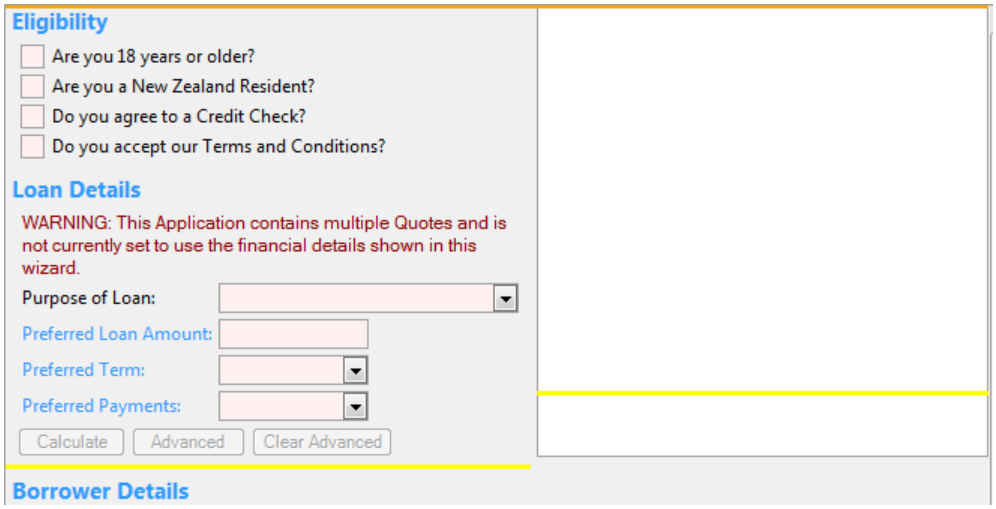
- Each 'Qty' Page Object has its Caption Width set to 0 rather than to Auto-Size.
  - If set to Auto-Size, the caption (even although it is blank) would be sized to match the widest Page Object caption, i.e., 'Client:'.
- Each of the 'Description' and 'Price' Page Objects are configured to 'Keep on the same line as the previous Page Object'.
- The 'Total' NumberBox (at the bottom of the 'Price' column, and selected in the above screenshot) is configured to 'Left Align With' one of the 'Price' Page Objects.
  - Left aligning with a 'Price' Page Object ensures that the Page Objects line up nicely.

Using Column Page Objects

For 'Flow' layout Pages, the following types of Page Object can be used to achieve a multi-column Layout:

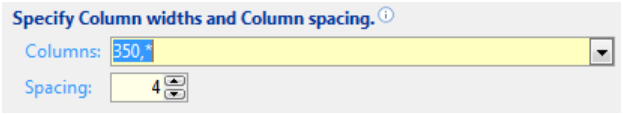
- Columns Start
  - Denotes the start of a multi-column section.
  - Defined the number of columns in the section and, optionally their widths.
- Column Break
  - Denotes a column break (i.e., the end of a particular column) within a multi-column section.
  - The number of 'Column Break' Page Objects should match the number of columns defined in the 'Columns Start' and, once the final 'Column Break' has been reached, the multi-column section is ended.

The following example shows a column of Page Objects to the left and a right-hand column containing only an HTML Panel which, in this case would be to display a financial summary from the figures entered to the left:



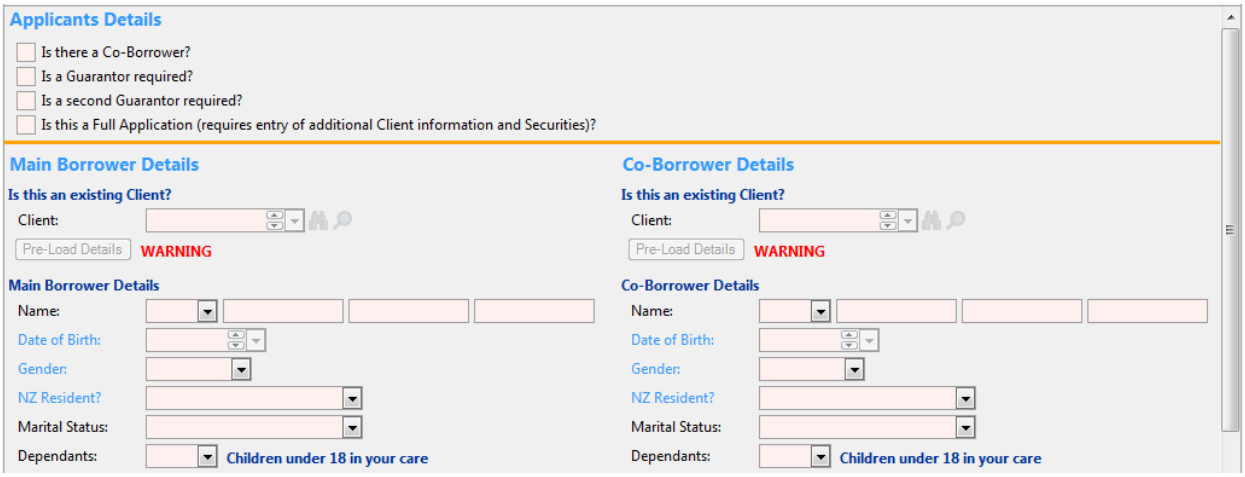
In this example:

- The 'Columns Start' is indicated as a horizontal orange line spanning the width of the page.
- Each 'Column Break' is indicated as a horizontal yellow line spanning the width of the column.
  - The second column break overlaps the HTML Panel since the HTML Panel's height is specified as 252 but it is set to 'Size to fill empty space on the Page' which, when the Page Object resides in a column, means that it will size to fill the entire column.
- The 'Columns Start' Page Object is configured as follows:



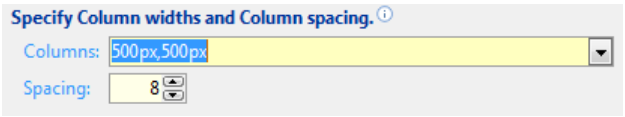
The makes the first column 350 pixels wide and the second column will fill the remaining Page width. A spacing of 4 pixels separates the columns.

The next example shows how you can arrange two Clients' (or Applicant) details next to each other:



In this example:

- The 'Columns Start' is indicated as a horizontal orange line spanning the width of the page.
- 'Column Break' Page Objects are not visible since the columns a taller than the screenshot.
- The 'Columns Start' Page Object is configured as follows:



The makes both columns 500 pixels wide with a gap of 8 pixels between columns.

Advanced Scripting

Script Objects

The entire Page Set has access to a special Page Set Handler object, accessed via the psh property. The `Initialise` method of the Page Set creates shortcuts to the UI and Reports objects as follows (from the template Script).

```
' Reporting and User Interface Objects
Public mReports As ISfinReports
Public mUI As ISUserInterfaceBL

Public Overrides Function Initialise() As Boolean

  ' Assume Success
  Initialise = True

  ' Initialise
  mReports = DirectCast(psh.Reports, ISfinReports)
  mUI = DirectCast(psh.UserInterface, ISUserInterfaceBL)

End Function
```

psh

The Page Set Handler (a finPageSetHandler type object) is the object which controls a Page Set.

This is accessed through the psh property which is defined on the Script class's base.

Some of the more commonly used Page Set Handler properties are:

- Pages
  - A collection of Pages.
  - This is typically used in a 'Tabbed Pages' or 'Single Page' type Page Set to show or hide pages, e.g.:

    ```
    psh.Pages("Individual").Visible = False
    ```
- CurrentPage, CurrentPageId, CurrentPageIndex
  - These are references to the current page.
  - Typically, these would only be used in a 'Tabbed Pages' or 'Single Page' type Page Set, e.g., to show a particular page you could do:

    ```
    psh.CurrentPageId = "Individual"
    ```
- FormHeadingColour
  - Allows the background colour of the form heading area to be changed.
  - This is specified as either a blank String (to use the default colour) or as an HTML style colour, e.g.:

    ```
    psh.FormHeadingColour = "red"
    ```
- FormTitle
  - Allows the Form Title (shown in the form heading area) to be overridden.
    - By default, this is set to the current Page's Description.
- NavigationMethod
  - This is a read-only property detailing the Page Set's Navigation Method.
  - Normally this would be the value that is defined on the Page Set but it can be overridden using an Application Shortcut or from an Account Application.

- A Page Set Script might check this in its `Initialise` method and, if displaying a 'Wizard' type Page Set as 'Tabbed Pages' (which is what Account Applications do once past the initial data capture phase), hide pages that are not applicable.

- ReadOnly
  - This indicates whether the Page Set is in read-only mode.
  - A Page Set Script should check this and never update underlying objects if it is True.

Some of the more commonly used Page Set methods are:

- FormClose()
  - Close the Page Set form.
- FormRefresh(formKey, recordId, newRecord)
  - Send a message to another finPOWER Connect form, asking it to refresh itself, e.g., if a Page Set has updated Account 'L10000', the following will refresh any open Accounts form currently displaying the record:

    ```
    psh.FormRefresh("Accounts", "L10000")
    ```

    NOTE: To see the Form Key of any finPOWER Connect form (apart from Modal forms), right-click the Form's tab and select **Special**, **Form Details**.
- GetBoolean(name), GetDate(name), GetString(name) etc
  - These methods get a Page Object's value given the Page Object name.
  - Generally, a Script would access the Page Object directly, e.g.,

    ```
    value = txtFirstName.Text
    ```

    - If, say, a Page Set has multiple versions of a particular Page Object with a similar name, e.g., txtFirstName1, txtFirstName2, txtFirstName3, the following code could access them within a loop, e.g.:

      ```
      For  i = 1 to 3
        value = psh.GetString("txtFirstName" & CStr(i))
      Next
      ```
- SetValue(name, value)
  - Set a Page Object's value based on its name, e.g.:

    ```
    psh.SetValue("txtFirstName1", "Dave")
    ```
- ValidatePageObjects(currentPageOnly)
  - Causes the Page Set to validate Page Objects as if the **OK**, **Next >** or **Finish** buttons have been clicked, i.e., beep and focus on any invalid Page Object (E.g., a mandatory TextBox with no text entered).

    ```
    If Not psh.ValidatePageObjects(False) Then
      mUI.MsgBox("Invalid!")
    End If
    ```

    NOTE: For Wizards, `currentPageOnly` will always be assumed to be `True`.

NOTE: Methods to show special forms such as the 'Financial Details' form are discussed in the Showing Special Forms section and those relating to Group Tags are discussed in the Group Tags section.

mUI

mUI (a ISUserInterfaceBL type object) is assigned in the Page Set Script's `Initialise` method and is a shortcut to a cut-down version of the User Interface functionality that finPOWER Connect User uses.

The most common use of mUI is to show message boxes and error messages using these methods of mUI:

- ErrorMessageShow()
  - Shows the latest error message from the finPOWER Connect business layer (or set from the Script via `finBL.Error.ErrorBegin`).
- MsgBox(prompt, buttons)
  - Shows a message box and returns the button clicked by the User.
  - The 'prompt' parameter is the message to display.
  - The 'buttons' parameter an Enum of buttons to show and the type of icon to display. These are bit flags, e.g., the following will cause a 'Question' icon to display along with 'Yes' and 'No' buttons of which the 'No' button is the default (focused) button:

    ```
    MsgBoxStyle.YesNo Or MsgBoxStyle.Question Or MsgBoxStyle.DefaultButton2
    ```

`mUI.CommonDialog` can be used to display modal forms. Most of the methods return a Boolean value where False indicates that the User has cancelled the dialog. Some of the more common methods are:

- GetFileOpen(returnFileName, defaultFileName, title, filter)
  - Shows the standard Windows 'File Open' dialog and has a `ByRef returnFileName` parameter which contains the selected file name.
  - The filter parameter restricts which files are listed as is formatted as per the following example (pipe separated values):

    ```
    All Files (*.*)|*.*|Excel Files (*.xls;*.xlsx)|*.xls;*.xlsx|PDF Files
    (*.pdf)| *.pdf
    ```
- GetInput(text, [title, heading, caption, maxLength, multiLine])
  - Shows a dialog into which the User can enter a text value.

`mUI.CommonForms` can be used to display non-modal forms. Some of the more common methods are:

- ShowAddressMap(address, [windowTitle, icon, showModal])
  - Shows an HTML viewer locating the supplied address (an ISAddressDetails object).
- ShowDataSetViewer(dataSet, [caption, formKey])
  - Shows a DataSet, e.g., the result of a database query.
- ShowHtmlViewer(html, [caption, formkey, icon, tempFileExtension, showModal])
  - Shows an HTML viewer to display the supplied HTML.

A sample of other useful methods available via mUI and its sub-objects is shown below:

Method	Description
mUI	
GetCodeDescriptionListFromEnquiryAction	Returns a list of records of the specified type displayed on other forms, e.g., specifying an 'actionId' parameter of 'GetAccounts' will return a list of Account codes displayed on other open forms. This is the mechanism used to present the User with a list of codes when right-clicking in a DBComboBox such as the Account box on the Account Close wizard.
mUI.CommonForms	
ShowDataSetViewer	Show a form to display a DataSet. This form allows the DataSet to be exported to various formats such as MS Word and Excel.
ShowHtmlViewer	Show a form to display HTML.
ShowAddressMap	Show a form to display an Address Map from the supplied `ISAddressDetails` object.
mUI.CommonDialog	
GetInput	Prompts to User with a popup box into which to enter a text value.
GetInputNumber	Prompts to User with a popup box into which to enter a number value.
GetListSelection	Prompts to User with a popup box from which to select a value from a list.

mReports

mReports (an ISfinReports type object) is assigned in the Page Set Script's `Initialise` method and is a shortcut to finPOWER Connect's reporting functionality.

This functionality can be used to run built-in Queries and Reports.

NOTE: Currently, only Queries can be manipulated via the business layer. Reports can only be run via Application Shortcuts and functionality is therefore very limited.

The following example runs the Account List query and shows the results:

```
Public Sub cmdRunQuery_Click(sender As Object,
                             e As finPageObjectClickEventArgs) Handles cmdRunQuery.Click

  Dim Query As ISQueryBase
  Dim Ok As Boolean

  ' Assume Success
  Ok = True

  ' Create Query
  Ok = mReports.CreateQuery(isefinQueryType.AccountList, Query)

  ' Execute Query
  If Ok Then
    ' Update Parameters
    With Query.Parameters
      .SetString("AccountTypes", "VL,CC,RC")
    End With

    ' Set Columns to include
    Query.Columns.SetIncluded("AccountId,Description,AccountTypeId,Balance,StatusText")

    ' Execute Query
    Ok = Query.Execute()
  End If

  ' Show Results
  If Ok Then
    mUI.CommonForms.ShowDataSetViewer(Query.DataSet, "Account List Results")
  End If

  ' Error
  If Not Ok Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

When using a Query from a Script, it is helpful to know the names of the parameters and columns that the Query uses. This is achieved as follows:

- Open the Run Query wizard (**Reporting**, **Run Query**).
- Right-click form tab and select **Special**, **Query and Report Details**.

This shows details about the Query's Parameters and Columns, e.g.:

Query

Query Id: **AccountList**

Parameters

Name	Type
General	
Select the Order By and Accounts to show.	
OrderBy	List
OrderByDesc	Boolean
Accounts	Range

Page Set Events

Many Page Set events are described elsewhere in this document. This section details with some of the more advanced, miscellaneous events.

PageSetActivate

This event fires when the Page Set is activated.

When this event fires depends on the type of Page Set and an `ActivateReason` event property can be used to determine why this event was called:

- **Single Page**, **Tabbed Pages**, **Wizard**
  - When the Page Set form is activated, e.g., when the User switches from another form to the Page Set form.
  - The following Activate Reasons apply to these types of Page Sets:
    - `FormActivate`
      - The Page Set form has been activated, e.g., it was not the active form but has now become the active form.
  - **NOTE:** This event will NOT fire when the Page Set form is first displayed. This is by design.
- **Inline Tabs**
  - Every time the Page Set is activated, e.g., in the Task Manager this would be every time the Folder displaying the Page Set is selected or, in the case of a the Task Manager 'Home Page' Page Set, every time the 'Home' tab is selected.
  - The following Activate Reasons apply to these types of Page Sets:
    - `FormActivate`
      - The Page Set form has been activated, e.g., it was not the active form but has now become the active form.
    - `FormRefresh`
      - The form hosting the Page Set has been refreshed, e.g., by the User pressing F5.
    - `TabActivate`
      - The tab (or page) on the form hosting the Page Set has been refreshed, e.g., in the Task Manager form the User has switched to the 'Home' tab.
  - **NOTE:** This event fires every time the Page Set is activated, including the first time it is activated. Also, for Task Manager Explorer folders that display a Page Set, only the `FromRefresh` reason is used.

This event can be used to achieve the following type of functionality:

- Refresh a Summary Page whenever this form is activated.
- Refresh a dropdown list whenever this form is activated.
  - For example, the Page Set may contain a 'Branch' DbComboBox based on the 'Branches' standard range.
    - By calling `cboBranch.DataSourceRefresh()` from the PageSetActivate event, the list will display any new Branches, e.g., if the User has the Branches form open and has just added a new Branch record and then re-activated the Page Set form.

WARNING: Attempting to perform intensive tasks, such as refreshing a list of Accounts, every time the Page Set form is activated may cause performance issues.

ActionNotification

Most built-in forms within finPOWER Connect support the concept of 'Notification Actions'.

For example, the Accounts form can response to an 'AccountLogsRefresh' notification by refreshing its Logs grid. This notification is sent to the form in many situations, e.g., upon saving a new Account Log record.

Using this event, a Page Set can response to this type of notification. However, most notification also receive 'Notification Data' which the recipient can use to decide what to do. For example, the 'AccountLogsRefresh' notification may contain a Key/ Value List detailing whether a Log has been deleted or the Id of the Account for which a new Log has been added.

The following table details some of the more common Notification Actions and when their Notification Data is likely to contain:

Action Id	Description	Notification Data

AccountLogsRefresh	An Account Log has been added, updated or deleted (from the Log form).	Nothing or an ISKeyValueList which may contain any of the following:
		`AccountId (String)`
		`LogPk (Integer)`
		`Added (Boolean)`
		`Deleted (Boolean)`
		`MultipleEffected(Boolean)`

ClientLogsRefresh	A Client Log has been added, updated or deleted (from the Log form).	Nothing of an ISKeyValueList which may contain any of the following:
		`ClientId (String)`
		`LogPk (Integer)`
		`Added (Boolean)`
		`Deleted (Boolean)`
		`MultipleEffected(Boolean)`

AccountRefresh	An Account has been added (via New wizard), updated or deleted.	Nothing or an ISKeyValueList which may contain any of the following:
		`AccountId (String)`
		`AccountPk (Integer)`
		`Added (Boolean)`
		`Deleted (Boolean)`

ClientRefresh	A Client has been added (via New wizard), updated or deleted.	Nothing or an ISKeyValueList which may contain any of the following:
		`ClientId (String)`
		`ClientPk (Integer)`
		`Added (Boolean)`
		`Deleted (Boolean)`

WARNING: Improper handling of Notification Actions may result in poor Page Set performance.

NOTE: Notifications are a mechanism built into the finPOWER Connect Windows User Interface; not the business layer. Notifications will only be fired where implemented and may not always be fired where you might expect.

This event can also be used to respond to Custom Form Actions. For example, a Summary Page displayed in an HTML Panel could contain an Application Shortcut hyperlink to call a special Action to add a Client Log, e.g.:

```
app://FormAction?action=AddClientLog&clientId=C10000&subject=My Subject
```

This will fire the Page Set's ActionNotification event passing in an `actionId` of 'AddClientLog' and `notificationData` as 'clientId=C100001&subject=My Subject'.

The Page Set can then handle this and parse the `notificationData` into a Key/ Value List to simplify retrieval of the parameters, e.g.:

```
Public Sub PageSet_PageSetActionNotification(sender As Object,
                                             e As
finPageSetHandlerPageSetActionNotificationEventArgs) Handles Me.PageSetActionNotification

  Dim kvl As ISKeyValueList

  Select Case e.ActionId
    Case "AddClientLog"
      ' Parse parameters
      kvl = finBL.CreateKeyValueList()
      If TypeOf e.NotificationData Is String Then kvl.FromUrlString(CStr(e.NotificationData))

      ' Validate
      If Len(kvl.GetString("clientId")) = 0 Then
        mUI.MsgBox("Client Id not specified.", MsgBoxStyle.Exclamation)
        Exit Sub
      End If

      ' Add Client Log
      Action_AddClientLog(kvl.GetString("clientId"), kvl.GetString("subject", "Default Subject"))
  End Select

End Sub

Private Sub Action_AddClientLog(clientId As String,
                                subject As String)

  Dim ClientLog As finClientLog

  ClientLog = finBL.CreateClientLog()
  With ClientLog
    ' Update
    .ClientId = clientId
    .Subject = subject

    ' Save
    If .Save() Then
      mUI.MsgBox(String.Format("Log added for Client '{0}'.", clientId),
                 MsgBoxStyle.Information)
    Else
      mUI.ErrorMessageShow()
    End If
  End With

End Sub

Public Sub cmdAddLog_Click(sender As Object,
                           e As finPageObjectClickEventArgs) Handles cmdAddLog.Click

  Action_AddClientLog("C10000", "My Subject")

End Sub
```

NOTE: This mechanism is similar to the CustomHyperlinkClick event that an HTML Panel can handle but is centralised and is not dependent on any particular HTML Panel handling it.

In the above example, both a hyperlink in an HTML Panel and a button (cmdAddLog) can be used to add a new Client Log.

The ActionNotification event can also be fired by the Page Set Handler (`psh`) using either of the following methods:

- `ExecuteAction(actionId, [notificationData])`
  - Fire the ActionNotification event optionally passing in notification data.
- `ExecuteActionKeyValueList(actionId, notificationData)`
  - Fire the ActionNotification event passing in notification data in the form of an `ISKeyValueList` object.

Showing Another Page Set

Sometimes it may be desirable to show another Page Set from a Page Set, e.g., to allow updating of a sub-object such as an Account Client from a Page Set which deals with entering an Account.

This is achieved in the following way:

- Use the `psh.FormShowPageSet()` method to open another, sub-Page Set.
  - Optionally, pass in the current Page Set's main object (E.g., a finAccount object) or a sub-object (E.g., finAccount.Clients(0)) as the `dataSource` parameter.
    - This is available to the sub-Page Set via `psh.DataSource`.
  - The sub-Page Set will be displayed pseudo-modally to the main Page Set.
- The sub-Page Set can then perform whatever action is necessary and then close itself.
  - The sub-Page Set can access the main Page Set via `psh.ParentPageSet`.
    - Generally there should be no reason to do this.
- The main Page Set then handles the PageSetFormClosed event and decides what to do.
  - This might be nothing if the sub-Page Set was cancelled or it might be to refresh certain details (E.g., a grid of Account Clients).

Showing Special Forms

The Page Set Handler can show various special forms.

These are shown via one of the `psh.FormShowXXX` methods. Apart from `psh.FormShowPageSet` (the previous section). The more common forms are described below.

Account Application Applicant

This form allows the editing of an existing or the addition of a new Account Application Applicant via the built-in New Applicant wizard.

NOTE: This form is only available if licensed for the Account Application Add-On.

The following example assumes that the Page Set Script has an `mAccountApp` variable which is a `finAccountApp` object. A grid type Page Object is being used to display the Applicants and this has a drilldown button and an 'Add' button:

```
Public Sub gridApplicants_ButtonClick(sender As Object,
                                      e As finPageObjectButtonClickEventArgs) Handles
gridApplicants.ButtonClick

  Select Case e.ButtonId
    Case "Add"
      If Not psh.FormShowAccountAppApplicant(mAccountApp) Then
        mUI.ErrorMessageShow()
      End If
  End Select

End Sub

Public Sub gridApplicants_RowDrilldown(sender As Object,
                                       e As finPageObjectRowDrilldownEventArgs) Handles
gridApplicants.RowDrilldown

  ' Edit Applicant
  If Not psh.FormShowAccountAppApplicant(mAccountApp, mAccountApp.Applicants(e.ListIndex)) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowAccountAppApplicant` method may return an error, e.g., if not licensed for the Account Application Add-On.

The `FormShowAccountAppApplicant` method has the following parameters:

- accountApp
  - A `finAccountApp` object.
- accountAppApplicant (optional)
  - The Applicant to edit or `Nothing` to add a new Applicant to the collection.

When this wizard is closed, the Page Set can use the 'AccountAppApplicantFormClosed' event. The following example refreshes a grid of Applicants when the form is closed:

```
Public Sub PageSet_AccountAppApplicantFormClosed(sender As Object,
                                                 e As
finPageSetHandlerAccountAppApplicantFormClosedEventArgs) Handles Me.AccountAppApplicantFormClosed

  ' Refresh Applicants Grid and select edited Applicant
  If Not e.Cancelled Then
    gridApplicants.VirtualDataRefresh(mAccountApp.Applicants.IndexOf(e.AccountAppApplicant))
  End If

End Sub
```

Account Application Collateral Item

This form allows the editing of an existing or the addition of a new Account Application Collateral Item via the built-in New Collateral Item wizard.

NOTE: This form is only available if licensed for the Account Application Add-On.

The following example assumes that the Page Set Script has an `mAccountApp` variable which is a `finAccountApp` object. A grid type Page Object is being used to display the Collateral Items and this has a drilldown button and an 'Add' button:

```
Public Sub gridCollateralItems_ButtonClick(sender As Object,
                                           e As finPageObjectButtonClickEventArgs) Handles
gridCollateralItems.ButtonClick

  Select Case e.ButtonId
    Case "Add"
      If Not psh.FormShowAccountAppCollateralItem(mAccountApp) Then
        mUI.ErrorMessageShow()
      End If
  End Select

End Sub

Public Sub gridCollateralItems_RowDrilldown(sender As Object,
                                            e As finPageObjectRowDrilldownEventArgs) Handles
gridCollateralItems.RowDrilldown

  ' Edit Collateral Item
  If Not psh.FormShowAccountAppCollateralItem(mAccountApp,
mAccountApp.CollateralItems(e.ListIndex)) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowAccountAppCollateralItem` method may return an error, e.g., if not licensed for the Account Application Add-On.

The `FormShowAccountAppCollateralItem` method has the following parameters:

- accountApp
  - A `finAccountApp` object.
- accountAppCollateralItem (optional)
  - The Collateral Item to edit or `Nothing` to add a new Collateral Item to the collection.

When this wizard is closed, the Page Set can use the 'AccountAppCollateralItemFormClosed' event. The following example refreshes a grid of Collateral Items when the form is closed:

```
Public Sub PageSet_AccountAppCollateralItemFormClosed(sender As Object,
                                                      e As
finPageSetHandlerAccountAppCollateralItemFormClosedEventArgs) Handles
Me.AccountAppCollateralItemFormClosed

  ' Refresh Collateral Items Grid and select edited Collateral Item
  If Not e.Cancelled Then

gridCollateralItems.VirtualDataRefresh(mAccountApp.CollateralItems.IndexOf(e.AccountAppCollateralI
tem))
  End If

End Sub
```

Account Financial

This form allows the editing of full Account financial details as per the Financial page of the New Account wizard.

This form would typically be used by a Page Set designed to enter an Account Quotation or an Account Application.

The following example assumes that the Page Set Script has an mAccount variable which is a finAccount object:

```
Public Sub cmdEditFinancial_Click(sender As Object,
                                  e As finPageObjectClickEventArgs) Handles cmdEditFinancial.Click

  If Not psh.FormShowAccountFinancial(mAccount) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowAccountFinancial` method may return an error, e.g., if the Account does not have sufficient information to allow editing of financial details.

The `FormShowAccountFinancial` method has the following parameters:

- account
  - A `finAccount` object.
    - In the case of Account Applications, this will generally be created on-the-fly using the `finAccountApp.CreateAccount()` method.
- forceAutomaticRecalculation
  - Indicates whether to force automatic recalculation to occur (i.e., the User does not have to click the 'Calculate' button) when editing financial details regardless of the User's preferences.
- allowCancel
  - Indicates whether to enable the 'Cancel' button. This should only be `True` when dealing with a temporary Account, e.g., an Account created on-the-fly from an Account Application.
- allowAccountTypeSelection
  - Indicates whether to allow Account Type Selection, i.e., whether the User can use the '< Back' button to move from the Financial page of the wizard to a page that allows the Account Type to be selected.

Account Payment Arrangement Add

This form allows adding of an Account Payment Arrangement (or Promise if the User does not have permission to add a Payment Arrangement).

The following example assumes that the Page Set Script has an mAccount variable which is a `finAccount` object:

```
Public Sub cmdEditFinancial_Click(sender As Object,
                                  e As finPageObjectClickEventArgs) Handles cmdEditFinancial.Click

  If Not psh.FormShowAccountPaymentArrangementAdd(mAccount) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowAccountAccountPaymentArrangementAdd` method may return an error, e.g., if the Account is not saved.

The `FormShowAccountFinancial` method has the following parameters:

- account
  - A `finAccount` object.

When this wizard is closed, the Page Set can use the 'AccountPaymentArrangementAddFormClosed' event, e.g., to refresh on-screen details:

```
Public Sub PageSet AccountPaymentArrangementAddFormClosed(sender As Object,
                    e As finPageSetHandlerAccountPaymentArrangementAddFormClosedEventArgs) Handles
Me.AccountPaymentArrangementAddFormClosed

  mUI.MsgBox(e.AccountId & ": " & e.Cancelled & ": " & e.AccountPaymentArrangementPK)

End Sub
```

Account Schedule

This form shows an Account Schedule.

This form would typically be used by a Page Set designed to enter an Account Quotation or an Account Application.

The following example assumes that the Page Set Script has an mAccount variable which is a `finAccount` object:

```
Public Sub cmdShowSchedule_Click(sender As Object,
                                 e As finPageObjectClickEventArgs) Handles cmdShowSchedule.Click

  If Not psh.FormShowAccountSchedule(mAccount) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowSchedule` method may return an error, e.g., if the Account does not have sufficient information to allow showing of an Account schedule.

The `FormShowAccountSchedule` method has the following parameters:

- account
  - A `finAccount` object.
    - In the case of Account Applications, this will generally be created on-the-fly using the `finAccountApp.CreateAccount()` method.
- accountCalc
  - An optional Account Calculation object. If omitted, the Account's current Calculation will be used.
- allowEdit
  - Indicates whether to enable editing functionality on the Account Schedule Form. If True, this will ignore the 'accountCalc' and use the Account's current Calculation.
- gotoWhatIf
  - Indicates whether to go to the first 'What If' Schedule entry, e.g., a 'What If' Account Payment.

Account Temp

This form shows an Account form for the supplied, temporary Account object, e.g., an Account generated from an Account Application.

The following example assumes that the Page Set Script has an `mAccount` variable which is a `finAccount` object:

```
Public Sub cmdShowAccount_Click(sender As Object,
                                e As finPageObjectClickEventArgs) Handles cmdShowAccount.Click

  If Not psh.FormShowAccountTemp(mAccount) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowAccountTemp` method may return an error, e.g., if the Account does not have sufficient information to allow showing via the Accounts form.

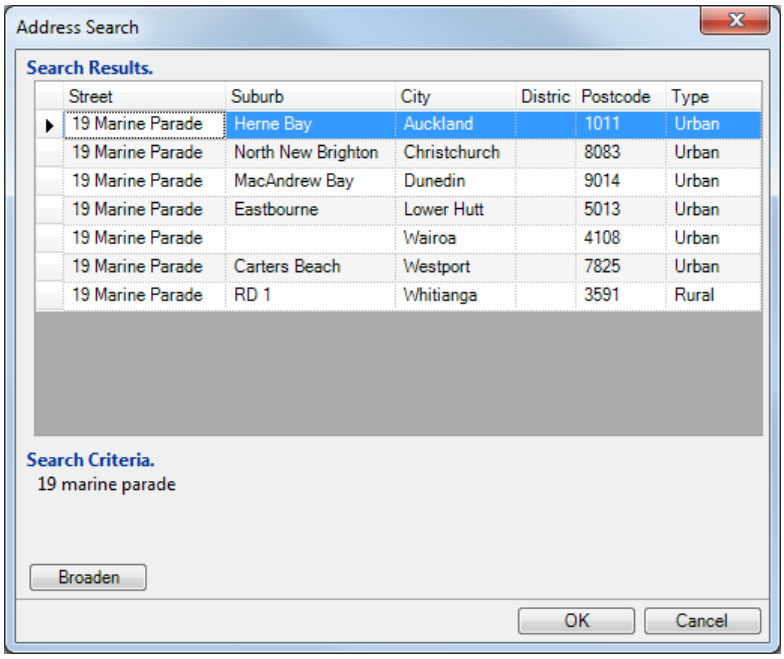
The `FormShowAccountTemp` method has the following parameters:

- account
  - A `finAccount` object.
    - In the case of Account Applications, this will generally be created on-the-fly using the `finAccountApp.CreateAccount()` method.
- heading
  - The form heading text or a blank String to use the default heading.
- summary
  - The form summary text or a blank String to use the default summary text.
- windowTitle
  - The window title text or a blank String to use the default window title.

Address Search

This form is a modal Address Search drilldown and is only available if licensed for the Addressing, Full Verification Add-On.

This form is used for searching on partial addresses and allowing the User to select the full address, e.g.:



The following code sample is run from a 'Find' button added to a TextBox:

```
Public Sub txtClientAddressStreet_ButtonClick(sender As Object,
                                              e As finPageObjectButtonClickEventArgs) Handles
txtClientAddressStreet.ButtonClick

  Dim AddressDetails As ISAddressDetails
  Dim SelectedAddressDetails As ISAddressDetails

  ' Build Address Details to Search
  AddressDetails = finBL.CreateAddressDetails()
  With AddressDetails
    .StreetAddressFull = txtClientAddressStreet.Text
    .Suburb = cboClientAddressSuburb.Text
    .City = cboClientAddressCity.Text
    .Postcode = cboClientAddressPostcode.Text
  End With

  ' Show Search Form
  If psh.FormShowAddressSearch(AddressDetails, SelectedAddressDetails) Then
    ' Update Address Fields
    If SelectedAddressDetails Is Nothing Then
      ' Cancelled
    Else
      With SelectedAddressDetails
        txtClientAddressStreet.Text = .StreetAddressFull
        cboClientAddressSuburb.Text = .Suburb
        cboClientAddressCity.Text = .City
        cboClientAddressPostcode.Text = .Postcode
      End With
    End If
  Else
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The TextBox only has a single button hence this sample does not have a `Select Case` to handle different buttons.

The `FormShowAddressSearch` method has the following parameters:

- addressDetails
  - An `ISAddressDetails` object.
    - This can be populated with as many or as few address details as required, e.g., you may choose to only set the `StreetAddressFull` property.
- addressDetailsSelected
  - The Address that the User selected in the Address Search form.

The method returns False if the User cancelled the Address Search.

Bank Account Enquiry Wizard

This form shows the Bank Account Enquiry wizard pseudo-modal to the Page Set form for either a Client or an Account Application Applicant.

```
Public Sub cmdEnquiry_Click(sender As Object,
                            e As finPageObjectClickEventArgs) Handles cmdEnquiry.Click

  If Not psh.FormShowClientBankAccountEnquiry(mClient) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowClientBankAccountEnquiry` and `FormShowAccountAppApplicantBankAccountEnquiry` methods may return an error, e.g., if for some reason the wizard cannot be displayed.

The `FormShowClientBankAccountEnquiry` method has the following parameters:

- client
  - A `finClient` object.
- serviceId
  - Optionally, the code of the Bank Account Enquiry Service to use.

The `FormShowAccountAppApplicantBankAccountEnquiry` method has the following parameters:

- accountAppApplicant
  - A `finAccountAppApplicant` object.
- serviceId
  - Optionally, the code of the Bank Account Enquiry Service to use.

The Page Set Script can handle to BankAccountEnquiryFormClosed event to decide what to do once the Bank Account Enquiry wizard has been completed of cancelled by the User, e.g.:

```
Public Sub PageSet_BankAccountEnquiryFormClosed(sender As Object,
                                                e As
finPageSetHandlerBankAccountEnquiryFormClosedEventArgs) Handles Me. BankAccountEnquiryFormClosed

  ' Bank Account Enquiry wizard completed
  If Not e.Cancelled Then
    ' Your code goes here
  End If

End Sub
```

Client Temp

This form shows a Client form for the supplied, temporary Client object, e.g., a Client generated from an Account Application.

The following example assumes that the Page Set Script has an `mClient` variable which is a `finClient` object:

```
Public Sub cmdShowClient_Click(sender As Object,
                               e As finPageObjectClickEventArgs) Handles cmdShowClient.Click

  If Not psh.FormShowClientTemp(mClient) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowClientTemp` method may return an error, e.g., if the Client does not have sufficient information to allow showing via the Clients form.

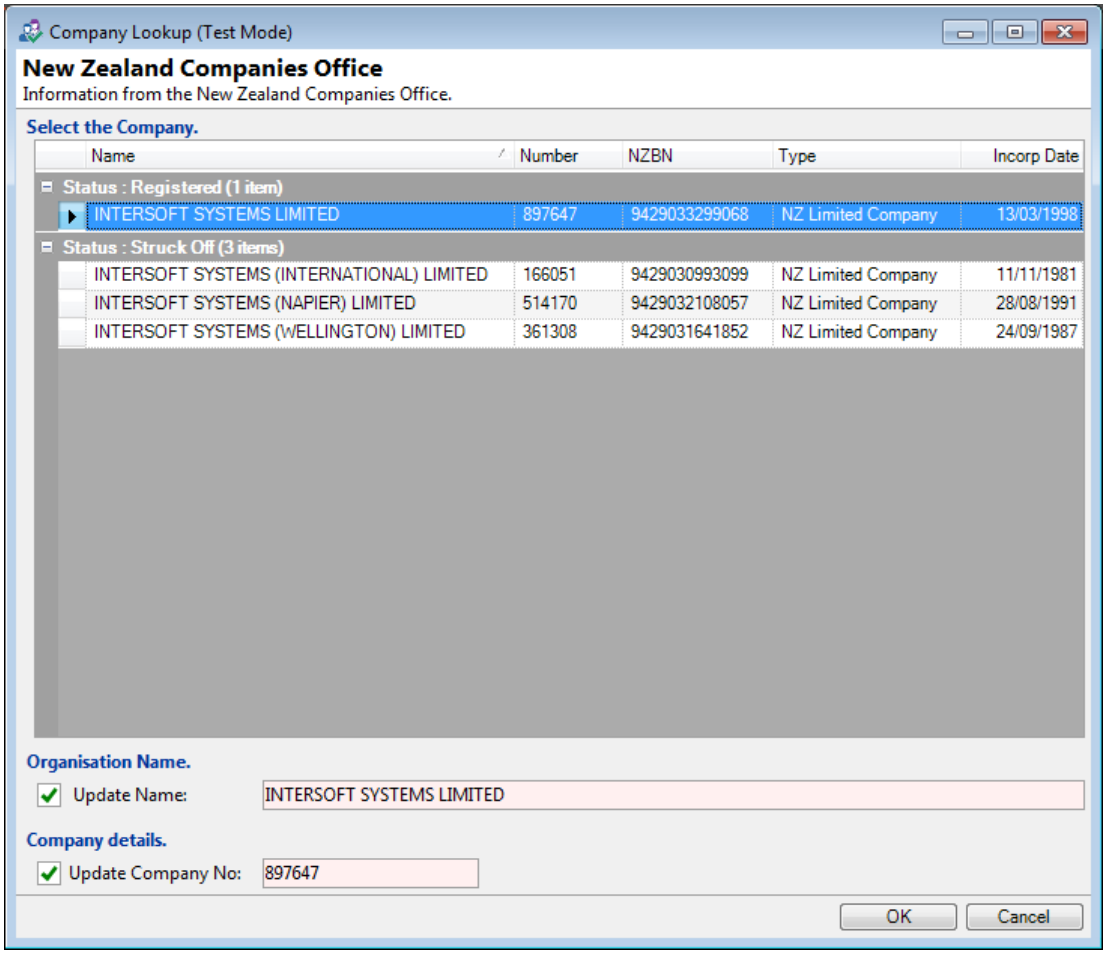
The `FormShowClientTemp` method has the following parameters:

- Client
  - A `finClient` object.
    - In the case of Account Applications, this will generally be created on-the-fly using the `finAccountAppApplicant.CreateClient()` method.
- heading
  - The form heading text or a blank String to use the default heading.
- summary
  - The form summary text or a blank String to use the default summary text.
- windowTitle
  - The window title text or a blank String to use the default window title.

Company Lookup

This form is a modal Company Lookup form and is only available if licensed for the Credit Enquiry Add-On and configured for a country that supports company lookups, e.g., Australia or New Zealand.

This form can be used to search for a company based on different search criteria, e.g., Company Name or Company Number, e.g.:



The following code sample is run from a 'Find' button added to two TextBoxes, txtOrganisationName and txtCompanyNumber:

```
Public Sub txtCompany_ButtonClick(sender As Object,
                                  e As finPageObjectButtonClickEventArgs) Handles
txtOrganisationName.ButtonClick, txtCompanyNumber.ButtonClick

  Dim ExistingDetails As ISKeyValueList
  Dim DetailsToUpdate As ISKeyValueList
  Dim Results As ISKeyValueList
  Dim SearchBy As isefinCompanyLookupSearchBy
  Dim SearchQuery As String

  ExistingDetails = finBL.CreateKeyValueList()
  With ExistingDetails
    .SetString("CompanyName", txtOrganisationName.Text)
    .SetString("CompanyNumber", txtCompanyNumber.Text)
  End With

  If sender Is txtOrganisationName Then
    SearchBy = isefinCompanyLookupSearchBy.CompanyName
    SearchQuery = txtOrganisationName.Text
  Else
    SearchBy = isefinCompanyLookupSearchBy.CompanyNumber
    SearchQuery = txtCompanyNumber.Text
  End If

  If psh.FormShowCompanyLookup(SearchBy, SearchQuery, Results, ExistingDetails, DetailsToUpdate)
Then
```

```
    If Results Is Nothing Then
      mUI.MsgBox("Cancelled by User.", MsgBoxStyle.Information)
    Else
      ' Update fields based on selected values (could use Results instead if we're not worried
about what the User selected)
      If DetailsToUpdate.Exists("CompanyName") Then txtOrganisationName.Text =
DetailsToUpdate.GetString("CompanyName")
        If DetailsToUpdate.Exists("CompanyNumber") Then txtCompanyNumber.Text =
DetailsToUpdate.GetString("CompanyNumber")
    End If
  Else
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The TextBoxes only have a single button hence this sample does not have a `Select Case` to handle different buttons.

The `FormShowCompanyLookup` method has the following parameters:

- searchBy
  - A `isefinCompanyLookupSearchBy` value to specify how the search is performed:
    - CompanyName
    - LegalName
    - BusinessNumber
    - CompanyNumber
- searchQuery
  - The value to search (based on searchBy), e.g., a Company Number.
- results
  - A key/ value list that is returned and contains details of the selected row in the Company Lookup.
- existingDetails
  - A key/ value list containing existing, known details. E.g., you might add entries for "CompanyName" and "CompanyNumber". The Company Lookup form would then display these along with a checkbox indicating that the User would like to update them (see the detailsToUpdate parameteer).
- detailsToUpdate
  - A key/ value list that is returned and contains details of any items on the form that the user has indicated that they would like to updated.

The method returns `False` if an error occurs or sets results to `Nothing` if the User cancelled the Company Lookup.

The `existingDetails`, `results` and `detailsToUpdate` key/ value lists accept/ return the following properties based upon the database country:

Country/ Service	Property	Type
Australia (ABN Lookup)	CompanyName	`String`
	CompanyNumber	`String`
	BusinessNumber	`String`
	LegalName	`String`
New Zealand (Companies Office)	CompanyName	`String`
	CompanyNumber	`String`

	CommencementDate	`Date`
	BusinessNumber	`String`
	LegalName	`String`

NOTE: The method, `finBL.CreditBureau.CanPerformCompanyLookup()` can be checked to determine whether company lookup functionality is available. This can be used to show/ hide lookup buttons.

Credit Enquiry Wizard

This form shows the Credit Enquiry wizard pseudo-modal to the Page Set form for either a Client or an Account Application Applicant.

```
Public Sub cmdEnquiry_Click(sender As Object,
                            e As finPageObjectClickEventArgs) Handles cmdEnquiry.Click

  If Not psh.FormShowClientCreditEnquiry(mClient) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowClientCreditEnquiry` and `FormShowAccountAppApplicantCreditEnquiry` methods may return an error, e.g., if for some reason the wizard cannot be displayed.

The `FormShowClientCreditEnquiry` method has the following parameters:

- client
  - A `finClient` object.
- serviceId
  - Optionally, the code of the Credit Enquiry Service to use.

The `FormShowAccountAppApplicantCreditEnquiry` method has the following parameters:

- accountAppApplicant
  - A `finAccountAppApplicant` object.
- serviceId
  - Optionally, the code of the Credit Enquiry Service to use.

The Page Set Script can handle to CreditEnquiryFormClosed event to decide what to do once the Credit Enquiry wizard has been completed of cancelled by the User, e.g.:

```
Public Sub PageSet_CreditEnquiryFormClosed(sender As Object,
                                           e As
finPageSetHandlerCreditEnquiryFormClosedEventArgs) Handles Me.CreditEnquiryFormClosed

  ' Credit Enquiry wizard completed
  If Not e.Cancelled Then
    ' Your code goes here
  End If

End Sub
```

New Client Wizard

This form shows the New Client wizard pseudo-modal to the Page Set form.

This form would typically be used by a Page Set designed to enter an Account Quotation.

```
Public Sub cmdAddClient_Click(sender As Object,
                              e As finPageObjectClickEventArgs) Handles cmdAddClient.Click

  If Not psh.FormShowNewClientWizard() Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowNewClientWizard` method may return an error, e.g., if for some reason it cannot be displayed.

The `FormShowNewClientWizard` method has the following parameters:

- client
  - An optional `finClient` object.
  - If supplied, this should be an unsaved Client.

The Page Set Script can handle to NewClientWizardFormClosed event to decide what to do once the New Client wizard has been completed of cancelled by the User, e.g.:

```
Public Sub PageSet_NewClientWizardFormClosed(sender As Object,
                                             e As
finPageSetHandlerNewClientWizardFormClosedEventArgs) Handles Me.NewClientWizardFormClosed

  ' New Client wizard completed
  If Not e.Cancelled Then
    ' Refresh Client DBCombo (to include new Client)
    cboClient.RefreshDataSource()

    ' Update Fields
    cboClient.Text = e.ClientId
  End If

End Sub
```

PPSR Search Wizard

This form shows the PPSR Search wizard pseudo-modal to the Page Set form for either a Client , an Account Application Applicant or an Account Application Collateral Item.

```
Public Sub cmdEnquiry_Click(sender As Object,
                            e As finPageObjectClickEventArgs) Handles cmdEnquiry.Click

  If Not psh.FormShowClientPPSRSearch(mClient) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowClientPPSRSearch`, `FormShowAccountAppApplicantPPSRSearch` and `FormShowAccountAppCollateralItemPPSRSearch` methods may return an error, e.g., if for some reason the wizard cannot be displayed.

The `FormShowClientPPSRSearch` method has the following parameters:

- client
  - A `finClient` object.

The `FormShowAccountAppApplicantPPSRSearch` method has the following parameters:

- accountAppApplicant
  - A `finAccountAppApplicant` object.

The `FormShowAccountAppCollateralItemPPSRSearch` method has the following parameters:

- accountAppCollateralItem
  - A `finAccountAppCollateralItem` object.

The Page Set Script can handle to PPSRSearchFormClosed event to decide what to do once the Credit Enquiry wizard has been completed of cancelled by the User, e.g.:

```
Public Sub PageSet_PPSRSearchFormClosed(sender As Object,
                                        e As finPageSetHandlerPPSRSearchFormClosedEventArgs)
Handles Me.PPSRSearchFormClosed

  ' PPSR Search wizard completed
  If Not e.Cancelled Then
    ' Your code goes here
  End If

End Sub
```

Security Statement Item

This form allows the viewing or editing of an existing Security Statement Item via the built-in Security Statement Item form.

NOTE: This form is only available if licensed for the Security Register Add-On.

The following example assumes that the Page Set Script has an `mSecurityStmt` variable which is a `finSecurityStmt` object and also a `mSecurityStmtItemNew` variable which is a `finSecurityStmtItemBase` object. A grid type Page Object is being used to display the Security Items and this has a drilldown button. The page also has a separate 'Add' button:

```
Public Sub cmdAdd_Click(sender As Object, e As finPageObjectClickEventArgs) Handles cmdAdd.Click

  ' Create new Item
  mSecurityStmtItemNew = mSecurityStmt.SecurityItems.CreateSecurityStmtItem("MV")

  ' Add to Security Statement
  mSecurityStmt.SecurityItems.Add(mSecurityStmtItemNew)

  ' Edit New Item
  If Not psh.FormShowSecurityStmtItem(mSecurityStmtItemNew, True) Then
    mUI.ErrorMessageShow()
  End If

End Sub

Public Sub gridSecurityItems_RowDrilldown(sender As Object,
                                          e As finPageObjectRowDrilldownEventArgs) Handles
gridSecurityItems.RowDrilldown

  ' Not Editing a new Item
  mSecurityStmtItemNew = Nothing

  If Not psh.FormShowSecurityStmtItem(mSecurityStmt.SecurityItems(e.ListIndex), True) Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: The `FormShowSecurityStmtItem` method may return an error, e.g., if not licensed for the Security Register Add-On.

The `FormShowSecurityStmtItem` method has the following parameters:

- securityStmtItem
  - A `finSecurityStmtItemBase` object.
- allowEdit
  - A Boolean value indicating whether to allow the item to be edited.

When this form is closed, the Page Set can use the 'SecurityStmtItemFormClosed' event. The following example refreshes a grid of Security Items when the form is closed:

```
Public Sub PageSet_SecurityStmtItemFormClosed(sender As Object, e As
finPageSetHandlerSecurityStmtItemFormClosedEventArgs) Handles Me.SecurityStmtItemFormClosed

  ' Remove New Item since Cancelled
  If e.Cancelled AndAlso mSecurityStmtItemNew IsNot Nothing Then
    mSecurityStmt.SecurityItems.Remove(mSecurityStmtItemNew)
  End If

  ' Refresh Grid
  If mSecurityStmtItemNew Is Nothing Then
    gridSecurityItems.VirtualDataRefresh(gridSecurityItems.ActiveDataRowIndex)
  Else

gridSecurityItems.VirtualDataRefresh(mSecurityStmt.SecurityItems.IndexOf(mSecurityStmtItemNew))
  End If

  ' Finalise
  mSecurityStmtItemNew = Nothing
```

```
End Sub
```

Running an Action Script

Action Scripts are typically written to be run from record-based forms such as the Accounts and Clients forms.

An Action Script can however be run from a Page Set using the `psh.ExecuteActionScript()` method as per the following example which loads a Client and calls an Action Script that may update the Client's Name:

```
Public Sub cmdTest_Click(sender As Object, e As finPageObjectClickEventArgs) Handles cmdTest.Click

  Dim Client As finClient
  Dim Ok As Boolean
  Dim RequiresRefresh As Boolean

  ' Assume Success
  Ok = True

  ' Initialise
  Client = finBL.CreateClient()

  ' Load Client
  Ok = Client.Load("paul")

  If Ok Then
    ' Show Details
    txtName.Text = Client.Name

    ' Execute Script
    Ok = psh.ExecuteActionScript("CCASENAME", "Client", Client, Nothing, RequiresRefresh)

    ' Has Action Script has updated the Target Object
    If Ok AndAlso RequiresRefresh Then
      Ok = Client.Refresh()

      ' Show Updated Details
      If Ok Then
        txtName.Text = Client.Name
      End If
    End If
  End If

  ' Error
  If Not Ok Then
    mUI.ErrorMessageShow()
  End If

End Sub
```

NOTE: If you don't supply a Script Id, the User will be prompted to select the Action Script to run based upon the supplied Target Object Type.

Parameters

Parameters can be passed to a Page Set in the following ways:

- By running the Page Set from an Application Shortcut.
  - Any parameters specified in the Application Shortcut will be available to the Page Set.
- By opening the Page Set from another Page Set using the `psh.FormShowPageSet()` method.
  - This method includes a `parameters` argument for passing parameters to the other Page Set.
- If the Page Set has been opened to create or edit an Account Application.
  - In the case of Account Applications, several parameters are passed automatically to the Page Set. These are details in the Page Sets section of the **finPOWER Connect 2 Account Applications** document.

Parameters can be accessed by the Page Set Script (typically in the `Initialise` method), e.g.:

```
AccountAppPk = psh.Parameters.GetInteger("accountAppPk")
AccountAppTypeId = psh.Parameters.GetString("accountAppTypeId")
```

Using Group Tags

Each Page Object can define a comma-separate list of Group Tags.

These can then be used to perform bulk operations, e.g., show or hide all Page Objects with a particular tag and also to affect the enabled, read-only and visible states of Page Objects without changing the Page Object's properties directly.

The following code assumes that a number of Page Objects have the tag 'CoBorrower'. These are then made visible/ invisible when a CheckBox named chkCoBorrower is changed:

```
Public Sub chkCoBorrower_Change(sender As Object,
                                e As finPageObjectChangeEventArgs) Handles chkCoBorrower.Change

  psh.GroupTagSetVisible("CoBorrower", chkCoBorrower.Value)

End Sub
```

Group Tag functionality works as follows:

- When the Page Set is first loaded, a collection of Group Tags (`psh.GroupTags`) is built.
  - This collection contains an item for each unique Group Tag defined by Page Objects.
    - E.g., if a Page Object defined Group Tags of 'CoBorrower,CoBorrowerPostalAddress', the collection would contain an item for 'CoBorrower' and another entry for 'CoBorrowerPostalAddress'.
- Each item in the Group Tags collection has the following properties:
  - `Enabled` (`True` by default)
  - `ReadOnly` (`False` by default)
  - `Visible` (`True` by default)
- Group Tags properties can be updated directly in the collection, e.g.:

  ```
  psh.GroupTags("CoBorrower").Enabled = False
  ```

  - Typically though, one of the GroupTag methods would be used, e.g.:

    ```
    psh.GroupTabSetEnabled("CoBorrower", False)
    ```
- Each Page Object has the following properties which are used to determine the state of a Page Object when displaying the Page Set.
  - `EnabledResolved`
    - Will return `True` if the Page Object's `Enabled` property is `True` AND ALL of the `Enabled` properties for the Group Tags for this Page Object are `True`.
  - `ReadOnlyResolved`
    - Will return `True` if the Page Object's `ReadOnly` property is `True` OR ANY of the `ReadOnly` properties for the Group Tags for this Page Object are `True`.
  - `VisibleResolved`
    - Will return `True` if the Page Object's `Visible` property is `True` AND ALL of the `Visible` properties for the Group Tags for this Page Object are `True`.

The following methods update Group Tag states.

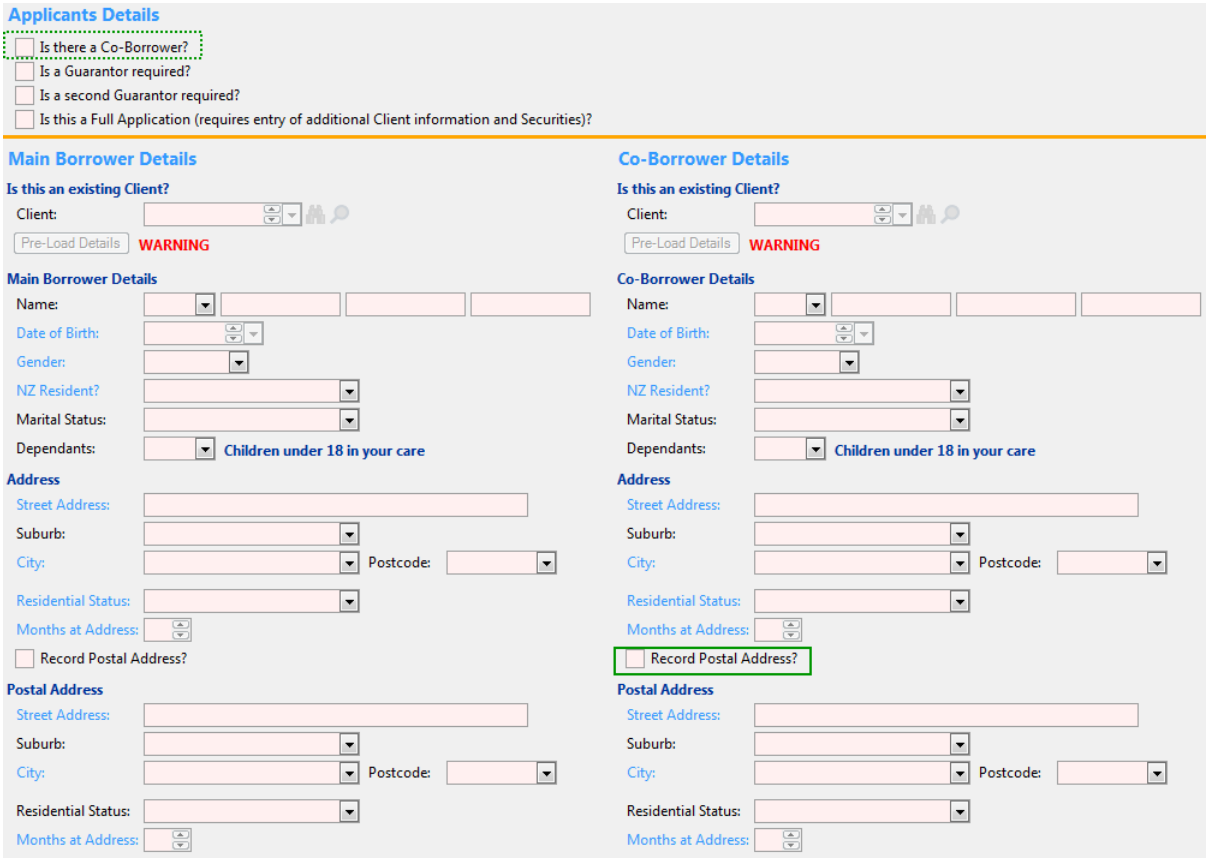
- GroupTagSetEnabled(groupTag, enabled)
  - Set the `Enabled` property of the specified tag.
- GroupTagSetReadOnly(groupTag, readOnly)
  - Set the `ReadOnly` property of the specified tag.
- GroupTagSetVisible(groupTag, visible)
  - Set the `Visible` property of the specified tag.

Multiple Tags

If is quite possible for a Page Object to list multiple, comma-separated tags.

If a Page Object does contain multiple tags, it will only be visible, enabled or non-read-only if ALL of its tags have been set to `True`.

The following is an example of where multiple tags might be used:



- This Page Set allows entry of co-borrower details but these are only displayed if a CheckBox named chkCoBorrower is checked.
  - All co-borrower Page Objects should have their Group Tags set to 'CoBorrower'.
- Within the co-borrower details, postal address details can be entered but these are only displayed if a CheckBox named chkCoBorrowerPostalAddress is checked.
  - All co-borrower Page Object relating to the postal address should have their group tags set to 'CoBorrower,CoBorrowerPostalAddress'.

The following code updates the Group Tags and therefore the visibility of the Page Objects as described above:

```
Public Sub chkCoBorrower_Change(sender As Object,
                                e As finPageObjectChangeEventArgs) Handles chkCoBorrower.Change

  psh.GroupTagSetVisible("CoBorrower", chkCoBorrower.Value)

End Sub

Public Sub chkCoBorrowerPostalAddress_Change(sender As Object,
                                             e As finPageObjectChangeEventArgs) Handles
chkCoBorrowerPostalAddress.Change

  psh.GroupTagSetVisible("CoBorrowerPostalAddress", chkCoBorrowerPostalAddress.Value)

End Sub
```

Other Group Functions

The following Page Set Handler (psh) methods are used to perform bulk updates:

- SetEnabledByGroupTag(groupTag, enabled)
  - Set the `Enabled` property of all Page Objects containing this tag.
- SetReadOnlyByGroupTag(groupTag, readOnly)
  - Set the `ReadOnly` property of all Page Objects containing this tag.
- SetVisibleByGroupTag(groupTag, visible)
  - Set the `Visible` property of all Page Objects containing this tag.

Application Shortcuts

A Page Set can be run from an Application Shortcut, e.g., a shortcut in the Task Pane or a hyperlink in an HTML Summary Page.

Examples of URL-based Application Shortcuts are:

```
PageSet?id=MyPageSet
PageSet?id=MyPageSet&pseudoModal=True
PageSet?id=MyPageSet&navigationMethod=TabbedPages&visiblePageCodes=General,Details
```

PageSet type Application Shortcuts can have the following parameters:

- id
  - A String value.
  - The code of the Page Set to run.
- visiblePageCodes
  - A String value.
  - A comma-separated list of Page Ids to display.
  - By default, the Page Set will show all Pages.
- navigationMethod
  - A String value based on the isefinPageSetNavigationMethod Enum.
  - Allows the Form Type (known internally as the Navigation Method) to be overridden, e.g., to display a 'Wizard' form as 'Tabbed Pages'.
  - Can be one of the following:
    - SinglePage
    - TabbedPages
    - Wizard
- modal
  - A Boolean value.
  - Forces the Page Set to be displayed as a modal form.
  - The Page Set can access the parent form via the 'ParentForm' parameter which will either be `Nothing` or an `ISFormBaseBL` object.
- pseudoModal
  - A Boolean value.
  - Forces the Page Set to be displayed as a pseudo-modal form, i.e., modal to its parent form only.
  - The Page Set can access the parent form via the 'ParentForm' parameter which will either be `Nothing` or an `ISFormBaseBL` object (see the next section).
- parentForm
  - An Object value.
  - Can be used to pass across the parent form in certain situations, e.g., from an Action-type Script (see the next section).

NOTE: Any other parameters specified in the Application Shortcut will be available to the Page Set Script via `psh.Parameters`.

Accessing the Parent Form

When a Page Set is launched from an Application Shortcut within another Page Set that uses either the modal or pseudoModal parameter, the Page Set can access the Parent Form as per the following example:

```
Dim ParentForm As ISFormBaseBL

' Get Parent Form
ParentForm = DirectCast(psh.Parameters.GetObject("ParentForm"), ISFormBaseBL)

' Refresh Parent Form (e.g., to reload Client record)
If ParentForm Is Nothing Then
  mUI.MsgBox("No parent form to refresh.", MsgBoxStyle.Exclamation)
Else
  ParentForm.Refresh()
End If
```

The `ISFormBaseBL` object has the following properties and methods:

- FieldsLoad method
  - Reload information from the form's Source object into the User Interface fields. This is useful if the Page Set has updated (but not saved) the 'Source' object.
- FormKey property
- FormRecordMode property
  - The Record Mode of the form, e.g., if the Clients form has a record loaded but is not in edit mode, this will be 'Loaded'
  - **NOTE:** For forms that do not support changing of records, e.g., the New Account wizard, this will always return 'Loaded'. Also, for Page Sets, many properties of the `ScriptInfo` object such as `FormKey` and `FormRecordMode` are not applicable and will just return default values.
- Refresh method
  - Causes the form to refresh, e.g., in the case of a record-based form such as the Clients form, this will cause the Client record to be reloaded.
  - This is useful if the Page Set has update and saved the underlying record.
- Source property
  - This is the source object for the form, e.g., a `finClient` object for the Clients form.
  - **NOTE:** Only a few forms currently expose a source object, therefore this property may by `Nothing`.
  - The following forms expose this:
    - Accounts form (`finAccount` object)
    - New Account wizard (`finAccount` object)
    - Clients form (`finClient` object)
    - New Client wizard (`finClient` object)
    - Security Statements form (`finSecurityStmt` object)

Action Scripts

Generally, the parentForm parameter is supplied automatically with the Application Shortcut but in the case of Action-type Scripts, it is possible to supply the parent form, e.g., the Accounts or Clients form from which the Application Shortcut is being run via a special ParentForm property that is available to the Action Script, e.g.:

```
Dim ParentForm As ISFormBaseBL

' Get Parent Form
ParentForm = ScriptInfo.Properties.GetObject("ParentForm")
```

```
' Execute Application Shortcut
Main = finBL.ExecuteApplicationShortcut(ApplicationShortcut, Nothing, Nothing, ParentForm)
```

Appendix A – Guidelines

Layout Guidelines

By following the style guidelines outlined in this section, a Page Set will look and act more like a built-in finPOWER Connect form.

General

Within finPOWER Connect, all forms work on a grid size of 4x4 pixels. This means:

- All Page Objects should have left/ right and top/ bottom padding that is a multiple of 4.
- Unless the Page Object's width and height are set to size automatically, they too should be a multiple of 4.
- If specifying a Caption Width (i.e., not using an automatically calculated Caption Width), this should be a multiple of 4.
- For Page Objects that support a 'TextBox Width', e.g., DBComboBoxes, ensure that this is a multiple of 4.
- Use 'Size to fill empty space on Page' to automatically expand Grids, HTML Panels and, if necessary, multi-line TextBoxes.

NOTE: You can view the actual sizes of the Page Objects by hovering over them in the Page Designer.

Page Object Widths

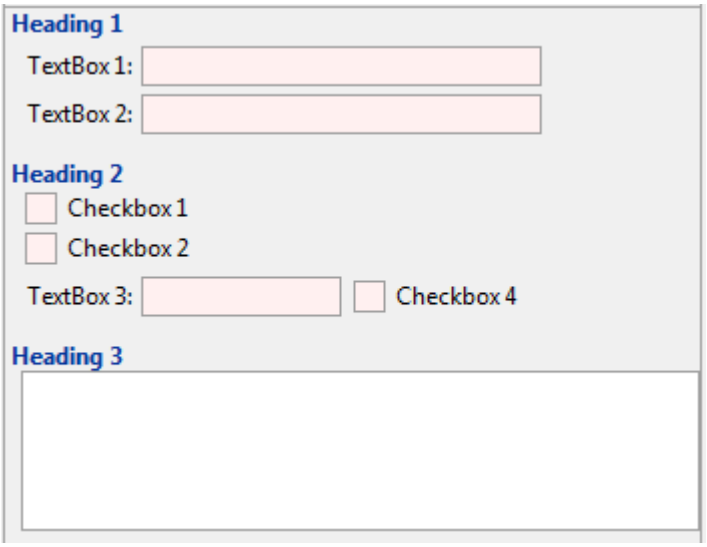
Use the following guidelines as a reference.

NOTE: Built-in buttons such as spin and dropdown (E.g., on a DBComboBox) are 16 pixels wide. Other buttons that appear after the Page Object are 20 pixels wide.

Type	Width	Notes
5 Char Code • With spin • With spin + dropdown	60 76 92	Short codes such as Branch Id.
10 Char Code • With spin • With spin + dropdown	100 116 132	General codes such as Account Id, Client Id etc.
20 Char Code • With spin • With spin + dropdown	180 196 212	Long codes such as User Id.
Date • With spin + dropdown	68 96	Read-only dates need not include spin and dropdown buttons unless it is more visually appealing. NOTE: Auto-sizing a DateBox's width will automatically use these values.
TextBox or **ComboBox**	8 x MaxLength	As a general rule, use this formula when creating or resizing a Page Object.
Button	80	Wider buttons are used on occasion, e.g., on Global Settings.

Page Object Spacing

The following Page Object spacing rules should be used to achieve the same look as built-in finPOWER Connect forms.

Page Objects	Notes
Heading 1 • Padding Top: 0 • Padding Left: 0 • Padding Bottom: 4 Padding for a section heading label to the next Page Object is 4 pixels.	
TextBox 1, TextBox 2 • Padding Top: 0 • Padding Left: 8 • Padding Bottom: 4 Page Objects within a section are left indented by 8 pixels. Padding of 4 pixels is used between Page Objects such as TextBoxes.	
Heading 2 • Padding Top: 8 • Padding Left: 0 • Padding Bottom: 0 A gap of 12 pixels is used between a TextBox type Page Object and the following section label. Since TextBox 2 has a bottom padding of 4 and Heading 2 a top padding of 8, this adds up to 12 pixels. Spacing between a section label and a Checkbox (or Grid or HTML Panel) is 4 pixels less than other Page Object Types hence a bottom padding of 0.	
Checkbox 1, Checkbox 2 • Padding Top: 0 • Padding Left: 0 • Padding Bottom: 0 Checkboxes are spaced vertically with a gap of 4 pixels less than TextBox type Page Objects hence Checkbox 1 has a bottom padding of 0.	
Checkbox 4 • Padding Left: 4	

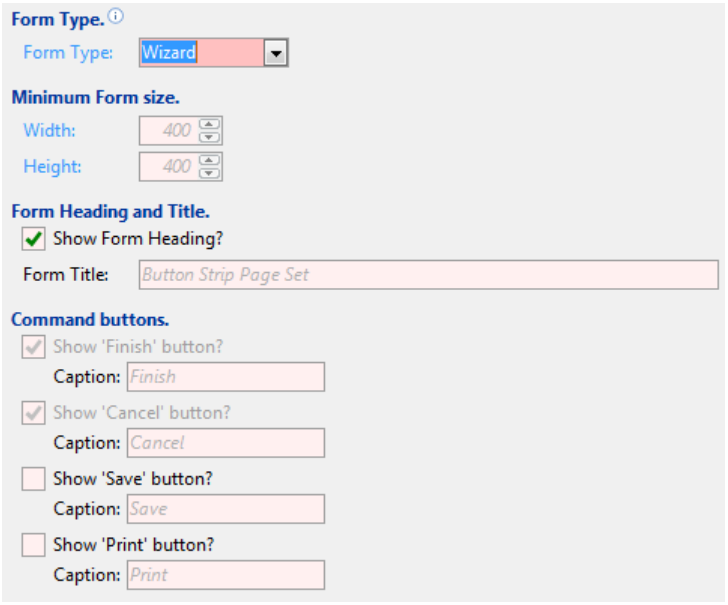
If a Checkbox appear after a TextBox type Page Object, the horizontal gap should be 4 pixels.	

Text Formatting and Other Guidelines

Text, e.g., section headings and captions are formatted to certain guidelines within finPOWER Connect. By following the guidelines in this section, a Page Set will look and act more like a built-in finPOWER Connect form.

General

The following is a screenshot of the Page Sets, Options page.



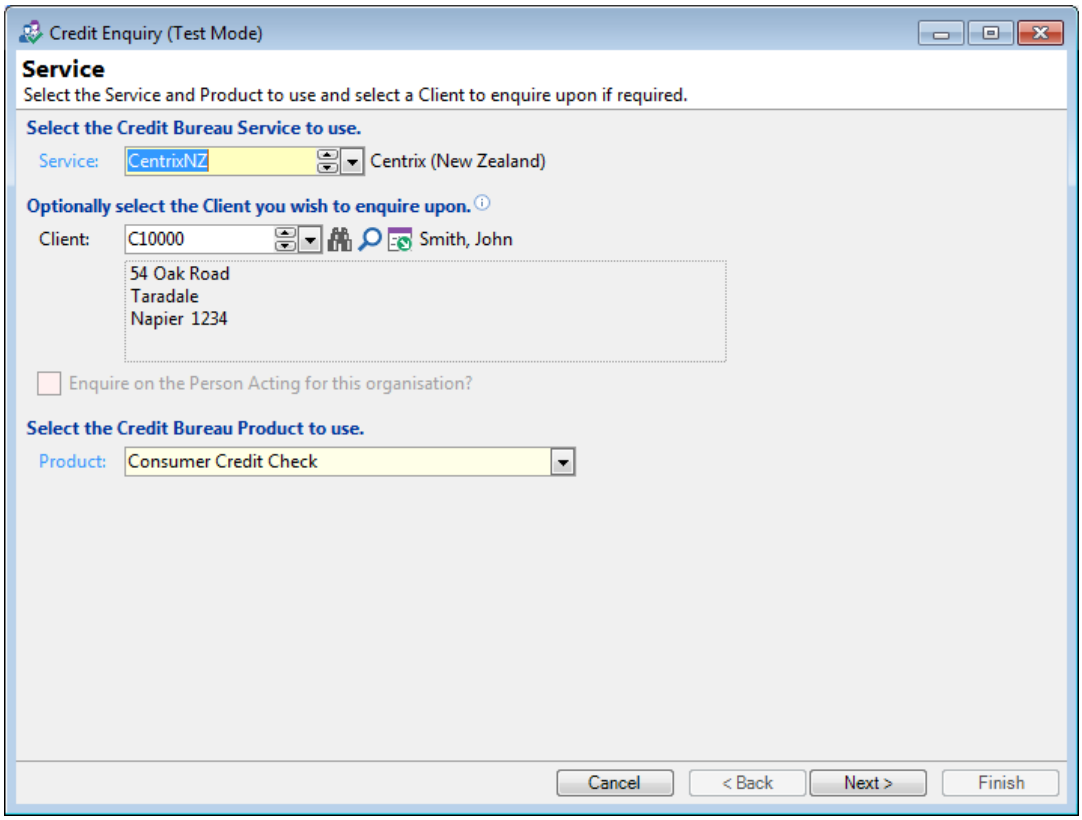
The following points should be taken into account when designing your own pages:

- Section headings, e.g., 'Minimum Form size.'
  - Use a Label with a style of 'Heading 4'.
  - Always end in a punctuation mark, generally a full stop.
  - Display an 'i' icon after their text is assigned a tooltip.
    - **NOTE:** Only 'Heading 4' style Labels do this and the icon will only be displayed when the Page Set is run, not in the Page Designer.
- Page Object captions should always end in a colon.
  - E.g., 'Form Type:'.
  - The exceptions are CheckBoxes.
- Use the 'Hint' property to provide a blank tip where a default applies.
  - E.g., the 'Form Title' above has a blank tip of 'Button Strip Page Set' to indicate that if this Page Object is left blank, this will be used as a default.
- Indent Page Objects that are directly related to the value of another Page Object.
  - E.g., the 'Caption' Page Objects beneath the CheckBoxes.
    - These are typically indented by 20 pixels.
- Disable Page Objects that are not applicable (set `Enabled = False`).
  - E.g., The 'Show Finish button' CheckBox is disabled if the 'Form Type' is set to 'Wizard' since Wizards always show a 'Finish' button.
- Use question marks for CheckBox captions if the caption is phrased as a question.

- E.g. 'Show Print button?'.

Wizards

The following is a screenshot of the Credit Enquiry wizard.



The following points should be taken into account when designing your own wizards:

- The Form Heading area displays the Page's Title and Summary.
  - The Title should generally be a very brief identifier for the wizard stage.
  - The Summary text should describe what the User is expected to do on this page.
- Section headings are generally phrased as instructions.
  - E.g., 'Select the Credit Bureau Service to use.'
- Typically, you would use the 'WizardValidate' event to perform any extra validation and prevent movement to the next page.
  - In the case of the Credit Enquiry wizard, the equivalent of the 'WizardValidate' event also loads details from the select Client that are used on the second page of the wizard. That way, if this process fails, the User cannot move past the first page.
    - In other situations, it may make more sense to use the 'WizardRefresh' event to load this data.

Appendix B – FormShow Application Shortcuts

Overview

Most forms within finPOWER Connect can be opened from an Application Shortcut.

Dragging the Form Heading area of a form to the Task Panel will create a 'FormShow' Application Shortcut providing the form supports being opened from an Application Shortcut.

Record-based forms, e.g., Documents, Branches, Accounts, Clients will create an application shortcut with the following parameters:

- form
  - The form's 'Form Key' which is how finPOWER Connect forms are identified.
- id
  - *Optional.* The Id of the record to load, e.g., a Client Id.
- page
  - *Optional.* The caption of the page to show on the form.
  - This can include a wildcard character on the end which is useful for pages such as the Logs page of the Clients form which shows a count of unactioned logs after the caption.

Dragging the Clients form to the Task Pane with Client C1000 loaded and displaying the Logs page will create the following Application Shortcut:

```
FormShow?form=Clients&id=C10000&page=Logs%2A
```

NOTE: In this example, the '%2A' on the end of 'Logs' is a URL encoded wildcard character (*).

There are a couple of ways of executing an Application Shortcut from Script code, executing a pre-formed URL, e.g.:

```
' NOTE: All parameters must be URL-encoded, e.g., using finBL.Runtime.HtmlUtilities.UrlEncode
finBL.ExecuteApplicationShortcutUrl("FormShow?form=Clients&id=C10000&page=Logs%2A")
```

Or using the ISApplicationShortcut object, e.g.:

```
' NOTE: Parameters are automatically URL-encoded
Dim ApplicationShortcut As ISApplicationShortcut

ApplicationShortcut = finBL.CreateApplicationShortcut()
With ApplicationShortcut
  .Action = "FormShow"

  With .Parameters
    .SetString("form", "Clients")
    .SetString("id", "C10000")
    .SetString("page", "Logs*")
  End With
End With

finBL.ExecuteApplicationShortcut(ApplicationShortcut)
```

URL-style Application Shortcuts can be embedded in HTML summary pages, e.g.:

```
<a href="app://FormShow?form=Clients&id=C10000&page=Logs%2A">Show Client</a>
```

The main advantages that using the ISApplicationShortcut object has over forming a URL directly are:

- All parameters are automatically escaped.
- Object type parameters can be included.
  - Some forms support Application Shortcut parameters that include Object, i.e., i.e., values that cannot be represented in a URL String.

NOTE: The `ISApplicationShortcut.ToUrlString()` method that returns a URL String, optionally including the `'app://'` prefix.

Many forms within finPOWER Connect can accept other parameters and these are detailed in the following sections (these will be built on over time).

Non-standard parameters that are likely to be of interest are bolded.

Execute Documents wizard

This is the form that is displayed to send a document to a range of Accounts, Clients etc via the Documents section in the Report Explorer.

Depending on which type of document you want to send, the Form Key (and the title) will vary between:

- DocumentsAccount
  - Account Documents
- DocumentsClient
  - Client Documents
- DocumentsSecurityStatement
  - Security Statement Documents
- DocumentsUser
  - User Documents

Application Shortcuts to show these types of form accept the following parameters:

- form
  - One of the Form Keys listed above.
- id
  - The Id of the Document to execute.
  - If supplied the first page of the wizard will be skipped.
- settingsUserDataPk
  - The primary key of the saved settings to load.
- **primaryRange**
  - A value for the primary range parameter, e.g., for Client documents, this should be a range of Client Ids, e.g., 'C10000,C10001'.
- **fileType**
  - Causes only Documents of the specified File Type to be listed in the Document dropdown.
    - Valid values are as per the `iseDocumentFileType` Enum:
      - Email
      - ExcelVba
      - Html
      - Log
      - Script
      - Sms
      - WordVba

A sample Application Shortcut URL is:

```
FormShow?form=DocumentsClient&primaryRange=C10000%2CC10001&fileType=Email
```

Appendix C – Samples

All samples in this section are designed to work with the finPOWER Connect demonstration databases.

Client Marketing Wizard (SMPL.CMK)

This presents a wizard into which criteria are entered and a grid of matching Clients is built.

The Page Set functions as follows:

- The User enters marketing criteria, e.g., the age range of Clients to list.
- Upon clicking the **Next >** button, a grid of matched Clients is displayed.
- All rows in the grid are 'Selected' by default but the User can unselect each row or use the buttons below the grid to select/ unselect ranges of Clients.
- The User can then use the buttons below the grid to send an Email, SMS or MS Word document to the selected Clients or, an 'Export' button to create a CSV file which could then be used for performing Mail Merges.

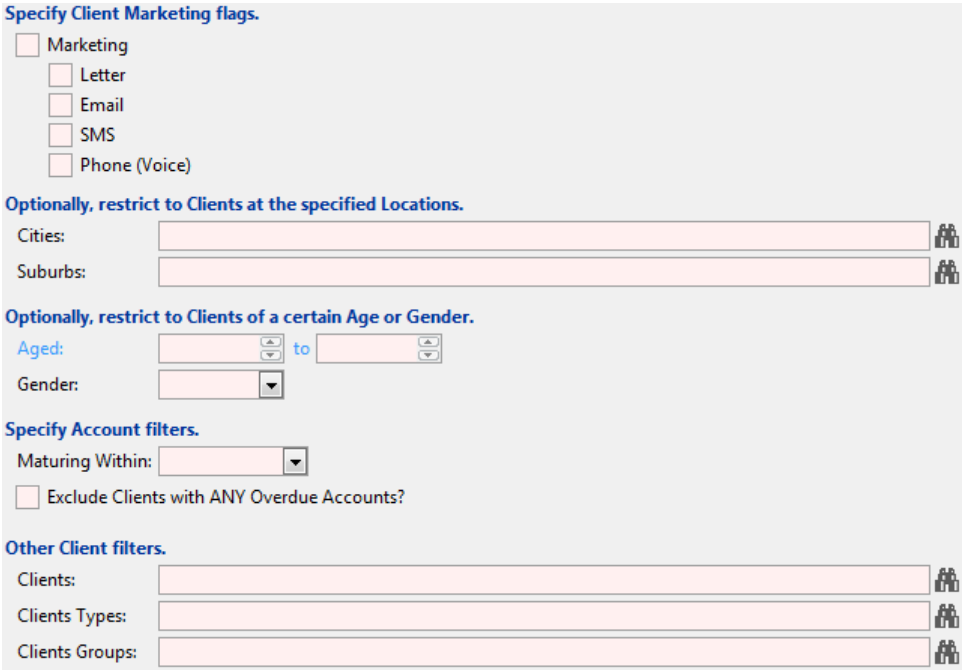
The main points this sample demonstrates are:

- A multi-page wizard type Page Set.
  - **NOTE:** Neither the WizardValidate or WizardMove events are used by this sample.
- Showing and hiding Page Objects based on licence settings.
  - Marketing flags differ based on whether the Advanced Clients Add-On is licensed.
  - The ability to send an SMS Document is only given if licensed for the SMS Add-On.
- Range Lookups.
  - Based on standard ranges, e.g., 'Client Types'.
  - Based on a database query, e.g., 'Cities' and 'Suburbs'.
- Creating and using a custom collection.
  - A collection (Generic List) of Marketing Clients is created and bound to the grid.
- Grids.
  - A drilldown column to open the Clients form.
  - A 'Selected' column that allows items in the grid to be checked as 'Selected'.
  - Grid buttons, including buttons to select rows and check/ uncheck the selected rows.
- Using the `finBL.StatusXXX()` methods to display a progress window.
- Using Application Shortcuts to show another form.
  - When sending Email, SMS or an MS Word Document, the range of selected Clients is passed across to the Document Execute wizard.

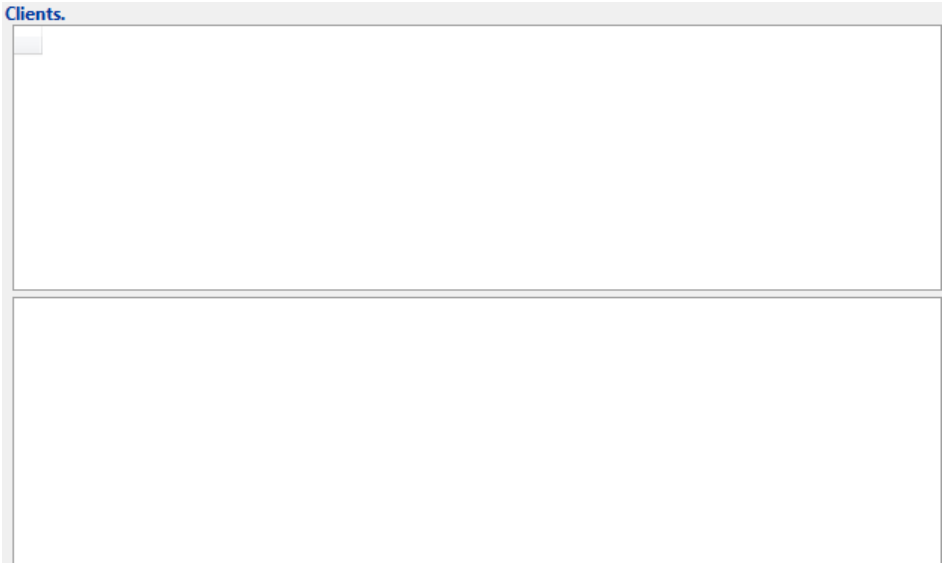
Pages

This Page Set consists of 2 pages:

- **FILTERS**
  - Entry of filter criteria.



- **CLIENTS**
  - A grid of Clients matching the filter criteria.



  - An HTML summary of the active grid row.
  - Both the Grid and the HTML Panel have their height configured to 'Size to fill empty space on Page'.

Functionality

When the Page Set is first run:

- **Initialise** event
  - Configures the grid (gridClients).
  - Creates a collection to bind to the grid (`mMarketingClients`).
  - Hides Page Objects not licensed.
  - Calls the `Form_Reset()` method.
- **Form_Reset** method
  - Sets Page Object defaults.
  - Clears the `mLastRefreshSql` variable (used to avoid re-querying the database unnecessarily).

When the User clicks the **Next >** button after entering details on the FILTERS page:

- **PageSet_WizardRefresh** event
  - Calls the `gridClients_Refresh()` method.
- **gridClients_Refresh** method
  - Uses an ISSelectQueryBuilder to create an SQL SELECT query based on the parameters entered on the FILTERS page.
  - Compares the SQL to the `mLastRefreshSql` variable and, if they are different:
    - Records the SQL in `mLastRefreshSql`.
    - Displays a progress window using the `finBL.StatusProgressBegin()` method.
    - Queries the database using sqb.ExecuteDataTable() so that the count of results can be accessed.
    - Iterates the results, adding an entry to the `mMarketingClients` collection if necessary.
      - Deceased Individuals are excluded. This could have been done in the SQL but is performed here as an example of post-processing the Clients before adding them to the collection.
      - The constructor of the `MarketingClient` class extracts details such as the current address and phone number from the supplied `finClient` object as sets properties on the class.
    - Updates the progress window using `finBL.StatusProcess()` method.
      - If the User has cancelled the process by clicking the 'Cancel' button in the progress window, the loop is exited.
    - Once the loop has finished, the `finBL.StatusProcessEnd()` method is called to close the progress window.
  - The Grid is refreshed passing in a value of zero to activate the first row.
  - Calls the `htmlPreview_Preview()` method to view a summary of the Client.
- **htmlPreview_Preview** method
  - Uses the built-in Client summary page to preview the active row in the grid.
    - The `finClient` object for the active row is passed to the `finBL.ScriptFunctions.BuiltInClientGeneralSummaryScript2()` method to generate the HTML.

The User then interacts with the grid (gridClients):

- **gridClients_InitialiseRow** event

    - This is called automatically by the Page Set for each item in the `mMarketingClients` collection when the grid is refreshed using the `gridClients.VirtualDataRefresh()` method.
    - Sets the display values for each grid cell.
        - Formats the address so that it is a single line address by replacing new line characters with commas.
- **gridClients_AfterRowActivate** event
    - Calls the `htmlPreview_Preview()` method to view a summary of the Client for the new active grid row.
    - Calls the `gridClients_ButtonsUpdate()` method to update the states of the buttons below the grid.
- **gridClients_ButtonsUpdate** method
    - Updates the states of the grid buttons, e.g., the Check/ Uncheck buttons are only enabled if one or more grid rows are selected.
- **gridClients_BeforeCellUpdate** event
    - The 'Selected' column is the only non-read-only column in the grid so this event will only fire when the User checks or unchecks the checkbox in this column.
    - The `Selected` property of the active item in the `mMarketingClients` collection is updated.
    - The Page Set then automatically calls the `gridClients_InitialiseRow` event for the row that has been updated.
- **gridClients_RowDrilldown** event
    - When the User clicks a drilldown button in the grid, this method executes an Application Shortcut URL to display the Clients form.
- **gridClients_ButtonClick** event
    - Handles clicking the buttons below the grid.
        - **Check** and **Uncheck** buttons
            - A list of selected rows is retrieved using the `gridClients.GetSelectedRowIndexes()` method and each corresponding entry in the `mMarketingClients` collection has its Selected property updated.
            - The grid is refreshed, passing in `-2` to preserve the active row and a value of `True` to preserve the selected rows.
        - **DocumentWordVba**, **Email** and **SMS** buttons
            - Builds a CSV list of Client Ids for each selected item in the `mMarketingClients` collection.
                - Items are excluded if there is no relevant value, e.g., no Email address when clicking the Email button.
            - Uses an application shortcut to show the Document Execute wizard (Client Documents). The primaryRange is specified as the range of Client Ids and the fileType is restricted, e.g., so the wizard will only allow selection of Email type Documents:

```
' Show Documents Execute form
ApplicationShortcut = finBL.CreateApplicationShortcut()
With ApplicationShortcut
  .Action = "FormShow"

  With .Parameters
    .SetString("form", "DocumentsClient")
    .SetString("primaryRange", List.ToCsvString())
    .SetString("fileType", iseDocumentFileType.Email.ToString())
  End With
End With
```

```
finBL.ExecuteApplicationShortcut(ApplicationShortcut)
```

- **ExportCsv** button
  - Prompts the User to enter a file name and then generates a CSV file.
    - The `ToCsvString()` method of `MarketingClient` is used to create a line. This uses the ISList object which ensures that all items are escaped and quoted correctly.
    - A default file name based on the current date and time and located in the User's Data Export folder is created.
- **Refresh** button
  - Clears the `mLastRefreshSql` variable and calls the `gridClients_Refresh()` method to rebuild the grid.
- **SelectAll**, **SelectNone**, **SelectUp** and **SelectDown** buttons
  - Use methods of the grid to change the selected rows.

IMPORTANT: When creating a SELECT query, it is important to respect User filters.

The method `gridClients_Refresh()` does this using the line:

```
.Append(finBL.CurrentUserInformation.FilterClientSqlWhere)
```

Similar filters exist for Accounts, Security Statements and Account Applications:

```
finBL.CurrentUserInformation.FilterAccountSqlWhere
finBL.CurrentUserInformation.FilterAccountAppSqlWhere
finBL.CurrentUserInformation.FilterSecurityStmtSqlWhere
```

Range Lookups

Several Range Lookups are shown on the FILTER page. These are simply TextBox type Page Objects with a 'Find' button. The ButtonClick event is used to display a Range Lookup form.

Many Range Lookups are based on standard finPOWER Connect ranges and are displayed as follows:

```
Public Sub txtRangeClientTypes_ButtonClick(sender As Object,
                                           e As finPageObjectButtonClickEventArgs) Handles
txtRangeClientTypes.ButtonClick

  psh.FormShowRangeLookupFromStandardRange(txtRangeClientTypes, isefinStandardRange.ClientTypes,
"")

End Sub
```

NOTE: The Client lookup (txtClients) has an extra `True` parameter at the end to force a proper Range Lookup form rather than the Client Search or Client List forms being displayed (based on User Preferences).

The Cities and Suburbs Range Lookups perform database queries. The Suburbs Range Lookup filters its list based on what is entered in the Cities lookup:

```
Public Sub txtRangeCities_ButtonClick(sender As Object,
                                      e As finPageObjectButtonClickEventArgs) Handles
txtRangeCities.ButtonClick

  Dim dt As DataTable
  Dim sqb As ISSelectQueryBuilder

  ' Build Query
```

```
  sqb = finBL.Database.CreateSelectQueryBuilder()
  With sqb
    .Table = "ClientContactAddress"
    .DistinctType = iseSelectQueryDistinctType.Distinct
    .Fields.Add("City")
    .SqlWhere.AppendComparisonNotNull("City")
    .OrderByFields.Add("City")
  End With

  ' Execute
  If sqb.ExecuteDataTable(dt) Then
    psh.FormShowRangeLookupFromDataView(txtRangeCities, dt.DefaultView, "City", "Cities")
  Else
    mUI.ErrorMessageShow()
  End If

End Sub

Public Sub txtRangeSuburbs_ButtonClick(sender As Object,
                                       e As finPageObjectButtonClickEventArgs) Handles
txtRangeSuburbs.ButtonClick

  Dim dt As DataTable
  Dim sqb As ISSelectQueryBuilder

  ' Build Query
  sqb = finBL.Database.CreateSelectQueryBuilder()
  With sqb
    .Table = "ClientContactAddress"
    .DistinctType = iseSelectQueryDistinctType.Distinct
    .Fields.AddList("Suburb,City")
    With .SqlWhere
      .AppendComparisonNotNull("Suburb")

      ' Restrict to City range
      If Len(txtRangeCities.Text) <> 0 Then
        .AppendRange("City", txtRangeCities.Text)
      End If
    End With
    .OrderByFields.Add("Suburb")
  End With

  ' Execute
  If sqb.ExecuteDataTable(dt) Then
    psh.FormShowRangeLookupFromDataView(txtRangeSuburbs, dt.DefaultView, "Suburb", "Suburbs")
  Else
    mUI.ErrorMessageShow()
  End If

End Sub
```

Add or Edit Client Form(SMPL.CLI)

This presents a single page form from which an existing 'Individual' type Client can be selected and edited or a new Client added.

This Page Set defines a number of Script constants to configure codes for Contact Methods and the Client Group and Client Type for new Clients.

The Page Set functions as follows:

- The User selects an existing Client using a DBComboBox (without a dropdown but with a search box) at the top of the page and pressing 'Enter' or clicking the 'Load Client' button.
  - Or, adds a new Client using the 'Add New Client' button.
- Client details can then be updated.
- The **Save** button can is used to update the Client record or the **Cancel Edit** button to cancel editing.

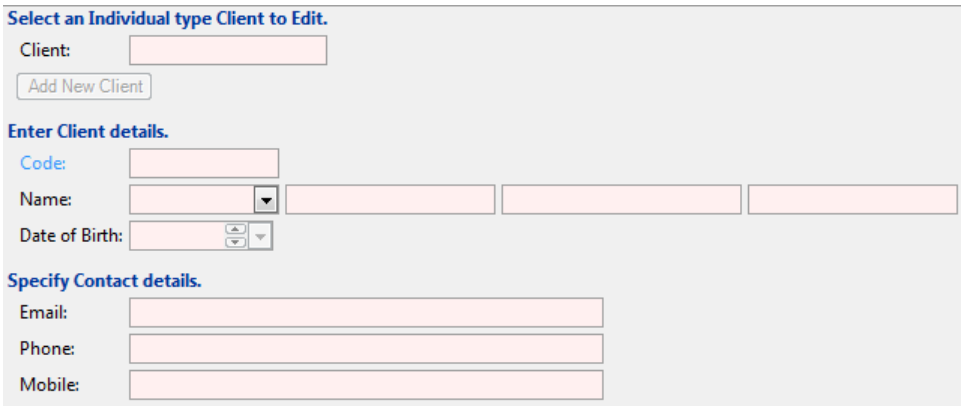
The main points this sample demonstrates are:

- A Single Page type Page Set.
- Using 'Quick Search' to locate a Client record.
- Enabling and Disabling Page Objects based on their Group Tags.
- Updating Command Buttons.
  - The captions on the **OK** and **Cancel** buttons are varied depending on whether a record is currently being edited.

Pages

This Page Set has a single Page.

- **CLIENT**
  - Entry of Client details.



  - The majority of this Page is occupied by a data entry form.
  - To make it easy to enable and disable this form, each of the Page Objects from the 'Enter Client details.' label down has a Group Tag of 'Client'.

Functionality

When the Page Set is first run:

- **Initialise** event
  - Picks up constants.
  - Creates an `mClient` object.
  - Adds buttons to the Client DBComboBox (cboLoadClient).
  - Calls the `Form_Reset()` method.
- **Form_Reset** method
  - Sets the `mEditing` flag to False and clears the cboLoadClient DBComboBox.
  - Calls the `Record_Load()` method to clear the form.
  - Focuses the cboLoadClient DBComboBox.
- **Record_Load** method
  - This method is called to load an existing Client or start editing a new Client record.
  - Loads or Clears the `mClient` object.
  - Validates that the loaded Client is an individual Client.
  - Calls the `Fields_Load()` method to populate the Page Objects from the `mClient` object.
  - Sets the `mEditing` flag and, based on this:
    - Enables or Disables Page Objects based on the 'Client' Group Tag.
    - Updates the captions on the Command buttons.
    - Sets focus.
- **Fields_Load** method
  - Populates Page Objects with general Client details (i.e., those that do not come from the `ContactMethods` collection).
  - Locates Email, Phone and Mobile Contact Methods.
    - Stores these in module-level variables.
    - Populates Page Objects with their values.

When the User loads an existing Client by entering a code in the Client DBComboBox and pressing Enter (or clicking the 'Load Client' button):

- **cboLoadClient_ButtonClick** event
  - Validates that a value is entered in cboLoadClient and beeps and focuses if not.
  - Calls the `Record_Load(False)` method to load an existing Client.

When the User adds a new Client using the 'Add New Client' button:

- **cmdAddNew_Click** event
  - Calls the `Record_Load(True)` method to load a new Client.

When the User has finished updating Client details and clicks the **Save** button:

- **PageSet_CommandButtonClick** event
  - Handles the `isefinPageSetCommandButton.Ok` button since this is actually the **Save** button.
  - Prevents the form from closing (the default action for both the **Cancel** and **OK** buttons) by setting `e.Cancel = True`.
  - Calls the `Record_Save()` method.

- **Record_Save** method
  - Calls the `Fields_Save()` method to update the `mClient` object.
  - If this is a new Client:
    - Sets the Client Group and Client Type.
  - Saves the Client record.
  - Displays a message to notify the User that the record has been updated.
  - Refreshes the cboLoadClient DBComboBox so that the new or updated Client appears in the list.
  - Resets the form by calling the `Form_Reset()` method.
- **Fields_Save** method
  - Populates general Client details from Page Objects (i.e., those that do not come from the `ContactMethods` collection).
  - Clears out any unsaved Contact Methods since these are recreated with each call to `Fields_Save()`.
  - Calls the `Fields_Save_ContactMethod()` method for each of the Email, Phone and Mobile Contact Methods.
- **Fields_Save_ContactMethod** method
  - This handles adding a new or updating an existing Client Contact Method.
  - If the passed in Client Contact Method is Nothing then this means that no Contact Method was found in `Fields_Load()`.
    - A new Client Contact Method is created (providing a value is specified).
    - This is inserted at the beginning of the `ClientContactMethods` collection since it makes sense for newer items to appear at the top of the list.
  - If the passed in Client Contact Method is not Nothing then an existing Contact Method is being updated:
    - If the passed in value is blank, then the User has cleared this Contact Method, therefore it is set as no longer current:
      - The `DateStop` property is set to yesterday's date.
        - This is necessary since this date is inclusive, therefore, setting it to today's date would indicate that the Contact Method is still current.
        - An alternative to this would be to set `DateStop` to today's date and set the `Historic` property to `True`.
    - If the passed in value is not blank, the Contact Method is updated and the `DateStop` property cleared to indicate that this Contact Method is current.

WARNING: The `finClientContactMethods.GetCurrentMobile()` and related methods are NOT used since they only returns contact methods that appear valid, e.g., if you enter a text value as a mobile phone number (such as 'N/A') and then used the above method, it will not return that Contact Method since it knows that mobile phone numbers must be numeric.

Custom Account Payment Wizard (SMPL.AP)

This presents a single-page wizard into which an Account Payment can be entered.

The Page Set functions as follows:

- The User selects an Account.
- The User enters Payment details and views a summary of how the Account will look once the Payment has been made.
- Upon clicking the **Finish** button, an Account Payment is made.

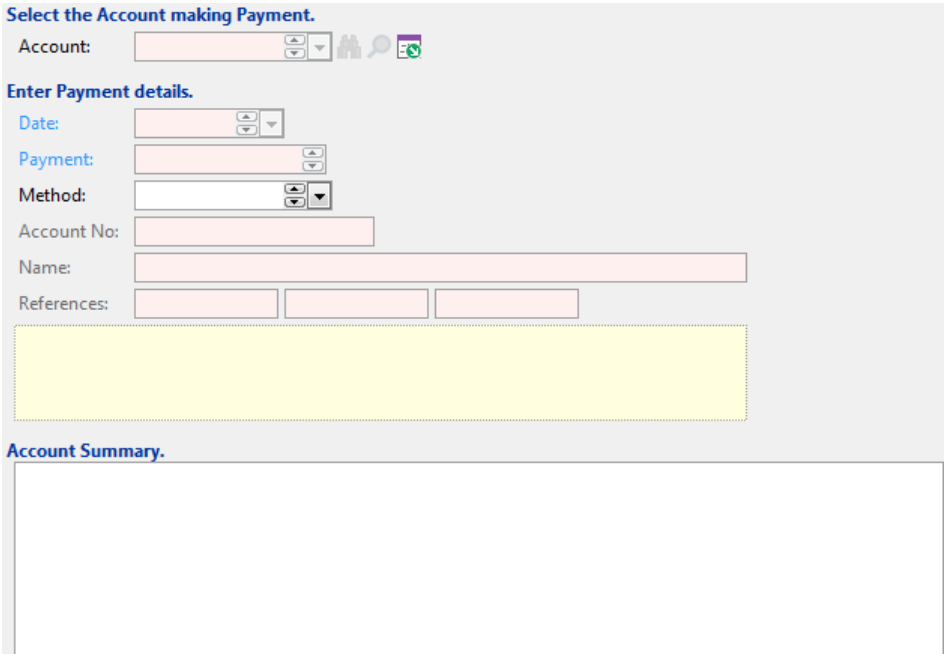
The main points this sample demonstrates are:

- A single-page wizard type Page Set.
  - **NOTE:** The WizardValidate event is used for demonstration purposes. This is only available to wizard type Page Sets.
- Maximum and minimum values on a Page Object.
  - The Payment NumberBox has these configured.
- Using a built-in Summary Page Script to display an HTML summary of the Payment.
- Using the Account Payment Details Page Object.

Pages

This Page Set has a single Page even though it is configured to be a wizard (since it demonstrates the PageSet_WizardValidate event).

- **PAYMENT**
  - Entry of Account Payment details.



  - This contains an Account Payment Details Page Object.
    - **NOTE:** A current limitation of this Page Object type is that it hard-codes it's Caption Width to 80. Therefore, all other Page Objects have their Caption Width set to 80 so they visually line up correctly.

Functionality

When the Page Set is first run:

- **Initialise** event
  - Picks up constants.
  - Creates an `mAccountPayment` object.
  - If the User's performance settings indicate that Account DBComboBoxes should not display a dropdown (thereby avoid a potentially large database read):
    - Sets cboAccount to use 'Fast Mode' which means that it will not show a dropdown list.
    - Reduces the width of cboAccount by 32 to account for it no longer displaying dropdown or spin buttons (each of which has a width of 16).
  - Calls the `Form_Reset()` method.
- **Form_Reset** method
  - Sets Page Object defaults.
  - Manually calls the `cboAccount_Change()` event
  - Sets focus.
- **cboAccount_Change** event
  - Either clears of loads the `mAccountPayment` object.
  - Updates the Payment details from those stored on the Account.
  - Calls the htmlAccountSummary_Preview() method to update the Account Payment preview.
- **htmlAccountSummary_Preview** method
  - Calls the `Fields_Save()` method to update the properties of the `mAccountPayment` object.
  - Uses the build in Account Payment Summary Script to produce an HTML Summary Page which is then used to update the htmlAccountSummary HTML Panel.
- **Fields_Save** method
  - Updates the properties of the `mAccountPayment` object from the corresponding Page Objects.

As the User enters data:

- **dateDate_Change**, **numPayment_Change**, **pdaPaymentDetails_Change** events
  - Calls the htmlAccountSummary_Preview() method to update the Account Payment preview.

When the User clicks the **Finish** button:

- **PageSet_WizardValidate** event
  - Since this is a wizard type Page, this event is called before the PageSet_CommandButtonClick event.
  - Validates that the Payment date is not in the future.
    - If it is, the User is alerted and `e.Failed` set to `True` to prevent this page from validating and therefore preventing the PageSet_CommandButtonClick event from firing.
- **PageSet_CommandButtonClick** event
  - Handles the `isefinPageSetCommandButton.Finish` button.
  - Prevents the form from closing (the default action for both the **Cancel** and **Finish** buttons) by setting `e.Cancel = True`.

- Calls the `Fields_Save()` method.
- Uses a message box to confirm that the User would like to make the Payment.
- Calls the `ExecuteCommit()` method of `mAccountPayment` to make a Payment.
- Notifies the User that the Payment was successfully made.
- Calls the `Form_Reset()` method to clear the wizard ready to make another Payment.

Simple Loan Quote Form (SMPL.LQ)

This presents a single page form from which allows a 'Quote' Account and, optionally, a new Client record to be created.

This Page Set defines a number of Script constants to configure codes for Contact Methods, Account Roles and Account Type.

The Page Set functions as follows:

- Displays a mainly disabled data-entry from.
  - Form is not enabled until all 'Eligibility' criteria are checked.
- Loan details can be entered and a preview of the repayment value is displayed.
- An existing Client can be selected or, optionally, details of a new Client entered.
- Upon clicking the 'Add Quote' button, a new Account is created and also a new Client if necessary.

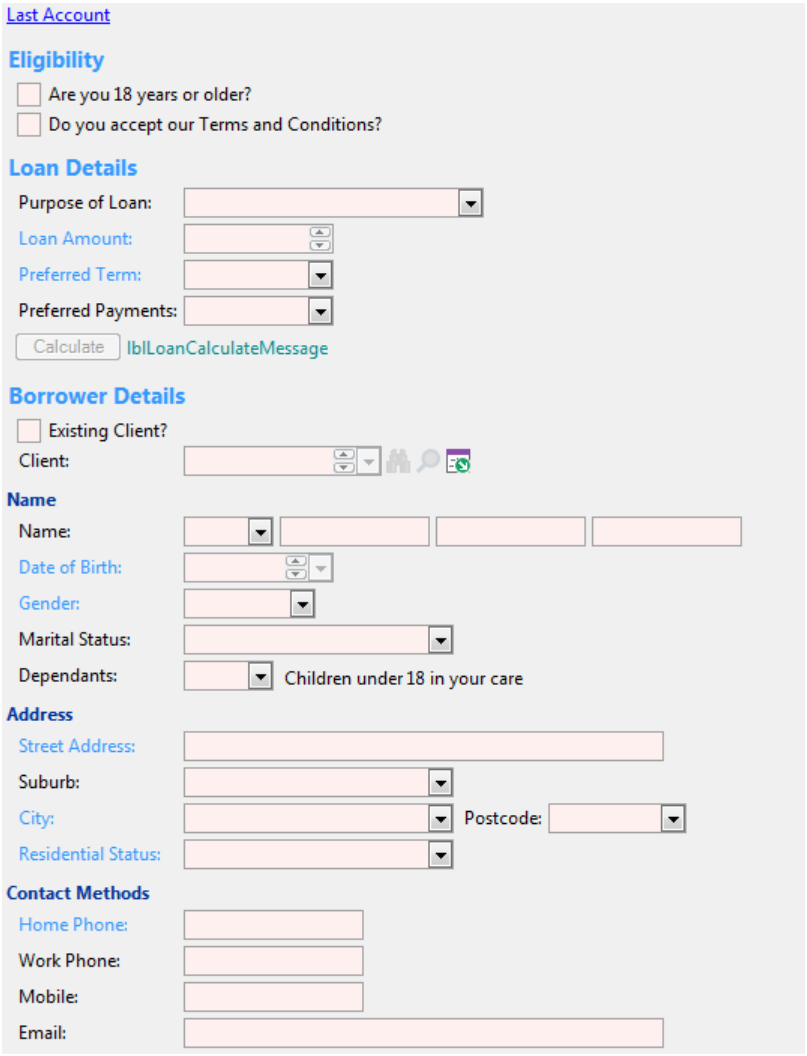
The main points this sample demonstrates are:

- A Single Page type Page Set.
- Using Group Tags to enable/ disable and show/ hide Page Objects.
  - The main data-entry fields are disabled until all eligibility CheckBoxes are checked.
  - Client Page Objects are hidden unless the 'Existing Client?' CheckBox is unchecked.
- The Payment calculation will occur automatically if User Preferences, Account, General, 'Automatically recalculate Account Financial figures' option is checked.
  - This can be overridden by the 'ForceAutoCalculate' constant.
- Addressing functionality.
  - A 'Find' button next to the 'Address' TextBox demonstrates the Address Lookup form.
  - The 'City' ComboBox presents a list of cities.
  - The 'Suburb' ComboBox lists suburbs for the selected city.
  - The 'Postcode' ComboBox lists postcodes for the selected suburb.

Pages

This Page Set has a single Page.

- **MAIN**
  - Quote capture page.



  - All Page Objects are disabled until both 'Eligibility' CheckBoxes are checked.
    - To achieve this, all other Page Objects include a Group Tag of either 'Loan' or 'Client'.
  - If 'Existing Client?' is checked, all Page Objects apart from the Client DBComboBox are hidden.
    - To achieve this, the Client DBComboBox includes a Group Tag of 'ClientExisting' and all other Client Page Objects, a Group Tag of 'ClientNew'.
  - The Address ComboBoxes all handle their Change events and update their lists on-the-fly.
    - The Suburb list is built based on the selected City.
    - The Postcode list is built based on the selected Suburb.

Functionality

When the Page Set is first run:

- **Initialise** event
  - Picks up constants.
  - Validates constants.
    - **NOTE:** Only validates whether they exist, not whether they are correct, e.g., you could enter a Phone Number type Contact Method for the Address.
  - Creates `mAccount` and `mClient` objects.
  - Calls the `Form_Reset()` method.
- **Form_Reset** method
  - Sets defaults for Page Objects that are not loaded from `mAccount` or `mClient`, e.g., the eligibility CheckBoxes.
  - Clears `mAccount` and `mClient`.
  - Calls the `Fields_Load()` method.
  - Calls various Change events and Refresh methods.
    - This ensures that all Page Objects are enabled/ disabled and shown/ hidden based on the defaults set at the top of this method.
    - The 'Refresh' methods, e.g., `cboClientAddressCity_Refresh()` ensure that Address Page Objects load their default lists.
  - Clears the message that appears alongside the 'Calculate' button.
  - Sets input focus.
    - Also scrolls to the top of the page since this is quite a long Page Set.
- **Fields_Load** method
  - Sets Page Objects from the `mAccount` object.
    - Only sets Purpose since this is defaulted from an information list when the Account is cleared.
  - Sets Financial Page Objects from `mAccount.Calculation`.
    - Calls the `Fields_LoadClient()` method to populate Client related Page Objects.
- **Fields_LoadClient** method
  - Populates Page Objects related to entry of a new Client.
  - This is separated from the Fields_Load() method to make it easy for the sample to be enhanced to pre-load details from an existing Client.

When the User has checked both 'Eligibility' CheckBoxes:

- **chkEligibility_Change** event
  - Enables the data-entry form using the `psh.GroupTagSetEnabled()` method.

As the User enters the 'Loan Details' section:

- **Financial_Change** event
  - Handles Change event for all financial-related Page Objects.
  - Calls the `CalculationInvalidate()` method.
  - If the Payments have changed, updates the Term since the business layer calculation may have changed this.
- **CalculationInvalidate** method
  - This method is called whenever any of the financial-related Page Objects is changed.

    - Decides whether to automatically perform an Account calculation or whether to just set a message 'Requires Calculation' to alert the User that they should click the 'Calculate' button to recalculate.
    - Calls the `CalculateInternal(False)` method if necessary.
- **CalculateInternal** method
    - Performs the Account calculation.
    - If no 'Payments' are entered, calculates 3 times:
        - Weekly
        - Fortnightly
        - Monthly
    - Calls the `CalculationError_Update()` method to either display a summary of the calculation or an error if the calculation failed.
    - Optionally, shows an error message dialog.
- **CalculationError_Update** method
    - Updates the label next to the 'Calculate' button (lblLoanCalculationMessage) to show either an error or a normal message (i.e., a summary of the calculation).
    - Changes the `LabelStyle` property to reflect whether an error is being displayed.

If the User clicks the 'Calculation' button:

- **cmdLoanCalculation_Click** event
    - Ensures a Term has been entered since a calculation cannot occur without this.
    - Calls the `CalculateInternal(True)` method.
        - `True` indicates to show the error dialog if necessary (not something we want to do if calculating automatically every time one of the financial Page Objects changes).

As the User enters the 'Borrower Details' section, they can either leave the 'Existing Client?' box unchecked and select an existing Client or, if they uncheck it:

- **chkClientExisting_Change** event
    - Hides the Client DBComboBox and shows all other Client Page Objects using the `psh.GroupTagSetVisible()` method.

As the User enters Address details:

- **cboAddressCity_Change** event
    - Calls the `cboClientAddressSuburb_Refresh()` and `cboClientAddressPostcode_Refresh()` methods to rebuild lists based on the selected City.
- **cboAddressSuburb_Change** event
    - Calls the `cboClientAddressPostcode_Refresh()` method to rebuild the list of postcodes selected City and Suburb.
- **cboClientAddressSuburb_Refresh** method
    - Uses the Address Interface (`finBL.Addressing`) to retrieve a list of suburbs for the specified City.
- **cboClientAddressPostcode_Refresh** method
    - Uses the Address Interface (`finBL.Addressing`) to retrieve a list of postcodes valid for the City and Suburb which are combined into an `ISAddressDetails` object.

When the User clicks the **Save Quote** button:

- **PageSet_CommandButtonClick** event
  - Handles the `isefinPageSetCommandButton.OK` button.
  - Prevents the form from closing (the default action for both the **Cancel** and **OK** buttons) by setting `e.Cancel = True`.
  - Calls the `Record_Save()` method.
- **Record_Save** method
  - Calls the `Fields_Save()` method to update both the mAccount and mClient objects.
  - Updates the `Name` and `Description` properties on `mAccount` using methods of `finAccount` to form default values.
  - Starts a database transaction.
    - Since we are saving (potentially) both a Client and Account record to the database.
  - Saves the new Client if necessary.
  - If a new Client is being created, updates `mAccount.Clients` to link to the new Client (something the `Fields_Save()` method could not do since no database record existed until now).
  - Saves the Account.
  - Commits the database transaction.
  - Updates and displays a hyperlink at the top of the page to show the Id of the new Account.
  - Resets the form by calling the `Form_Refresh()` method.