

SYSPRO 8

Application Designer

Overview Guide

Version: 007 (SYSPRO 8 2023)

Last Published: April 2024



SYSPRO Help and Reference

Copyright © 2024 SYSPRO Ltd

All rights reserved

No part of this document may be copied, photocopied, or reproduced in any form or by any means without permission in writing from SYSPRO Ltd. SYSPRO is a trademark of SYSPRO Ltd. All other trademarks, service marks, products or services are trademarks or registered trademarks of their respective holders.

SYSPRO Ltd reserves the right to alter the contents of this document without prior notice. While every effort is made to ensure that the contents of this document are correct, no liability whatsoever will be accepted for any errors or omissions.

This document is a copyright work and is protected by local copyright, civil and criminal law and international treaty. This document further contains secret, confidential and proprietary information belonging to SYSPRO Ltd. It is disclosed solely for the purposes of it being used in the context of the licensed use of the SYSPRO Ltd computer software products to which it relates. Such copyright works and information may not be published, disseminated, broadcast, copied or used for any other purpose. This document and all portions thereof included, but without limitation, copyright, trade secret and other intellectual property rights subsisting therein and relating thereto, are and shall at all times remain the sole property of SYSPRO Ltd.

WHAT'S CHANGED?

SYNOPSIS	READ MORE...
Create Transactional Data Sources	Transactional Data Sources
Create Web Service Data Sources	Web Service Data Sources
New Tag Cloud control	Tag Cloud control
Form Control improvements	Validate field on change How to enable custom form fields on FORMS for end-users How to show the Description of a 'key' item in a Form How to apply a tooltip (key description) to a field using a Data Source How to show multiple column names for a Form field
List view control improvements	Manipulating the listview in the Designer
Data Source improvements	Custom Data Sources
Manage Version History	Version history
Trace execution of an application	Application Designer Tracer
Avanti application improvements	Improving the Performance of an Avanti Application
Dialog Box improvements	How to use a Dialog Box

ANNOTATION

The contents of this document are subject to change.

Contents

Introducing the Application Designer	5
Benefits and features	5
Understanding the Application Designer	5
The Basics	6
Designing for Avanti	9
Creating a New Application	10
Designing a Toolbar.....	19
Designing a Context Menu	32
Creating Task Dialogs	33
Creating Avanti Web Views	41
Using Data Sources.....	50
Adding Custom Applications to the Desktop or Web UI Menu.....	104
Version history	106
Application Designer Tracer	108
Getting Help.....	112
Programming the Application.....	113
Application Designer VBScript Events	114
Applying a SYSPRO Callout in script	116
How to use a Dialog Box	118
Event Cycle Processing.....	120
Some Basics	120
Some VBScript Suggestions	123
Functions and Classes	123
Comment the Code	123
AppDesigner_OnUserEvent	124
The Use of Constants	124
Re-useable VBScript Functions and Classes	125
Debugging and Logging Progress	126
Using Persisted Variables.....	127
Passing Parameters to an Application	128
Passing execution back to the calling program	129
Retrieving Setup Options	130
Setting up the Datastore	131
Understanding the Controls.....	134
Chart Control	134
Gauge Control	153
List View Control	156
Form Control	170
Tree View Control.....	188
Syntax Editor Control	194

XAML Control.....	195
Notepad Control	213
Document Viewer	214
Image Viewer	216
Tile Control.....	217
Flowgraph Control	218
WebBrowser Control.....	220
User Control.....	221
Drop Files Control	222
Tag Cloud control.....	226
Working with Projects.....	228
Managing Projects	228
Creating New Projects	229
Maintaining Projects.....	230
Sample Apps.....	233
Appendix A – VBScript code snippets	234
App Comment section.....	234
Catching EventType	234
Displaying OnUserEvent Parameters	234
Get Local Windows Temp Directory Name.....	235
Toolbar Constants.....	235
Writing Text to a File – Simple Function.....	235

Introducing the Application Designer

The Application Designer is an application development environment that harnesses the SYSPRO User Interface, its business logic, its event handlers, and a Microsoft scripting language. It presents them in an easy-to-use interface which generates applications that run in SYSPRO.

It is designed to reduce the complexity of coding by providing a simple (low code) interface into all the SYSPRO functionality.

BENEFITS AND FEATURES

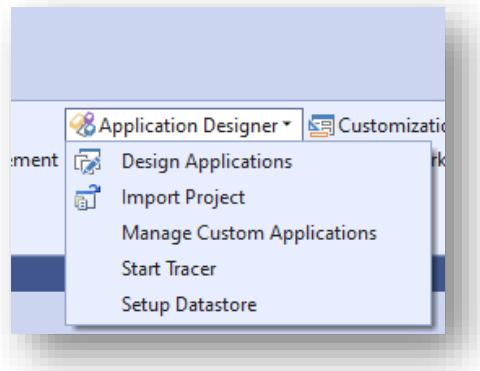
- Improved productivity as it allows developers to create a new UI for a program in a matter of minutes rather than hours.
- No additional technology required to produce and run applications.
- Improved storyboarding:
Because of the rapid time to create a UI, it can make it easy to produce a prototype and get sign-off approval for it.
- The complexity of program development is hugely reduced, as a single text file describes the entire application and can be easily deployed on a target machine.
- Multiple applications can be handled using Projects to deploy a complete solution to the end-user.
- Applications can be designed to run in both the SYSPRO Desktop and SYSPRO Web UI (Avanti).
- The Application Designer is not only used by partners and customers; it is also used by SYSPRO development to build standard SYSPRO applications. It is therefore, a robust, tried and tested platform for all types of applications.

UNDERSTANDING THE APPLICATION DESIGNER

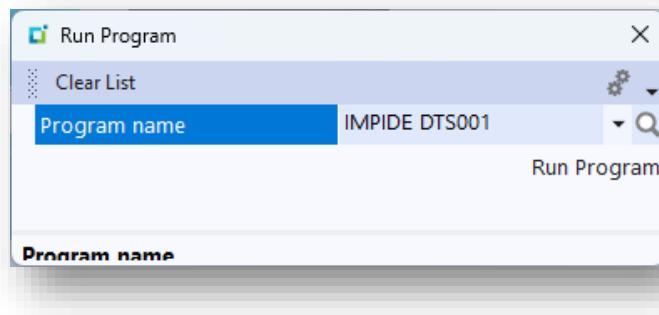
The designer is used to create applications to be used by end-users who in turn might wish to further customize the application by adding custom form fields, adding their own customized panes and a myriad other ways of customizing and personalizing the application.

THE BASICS

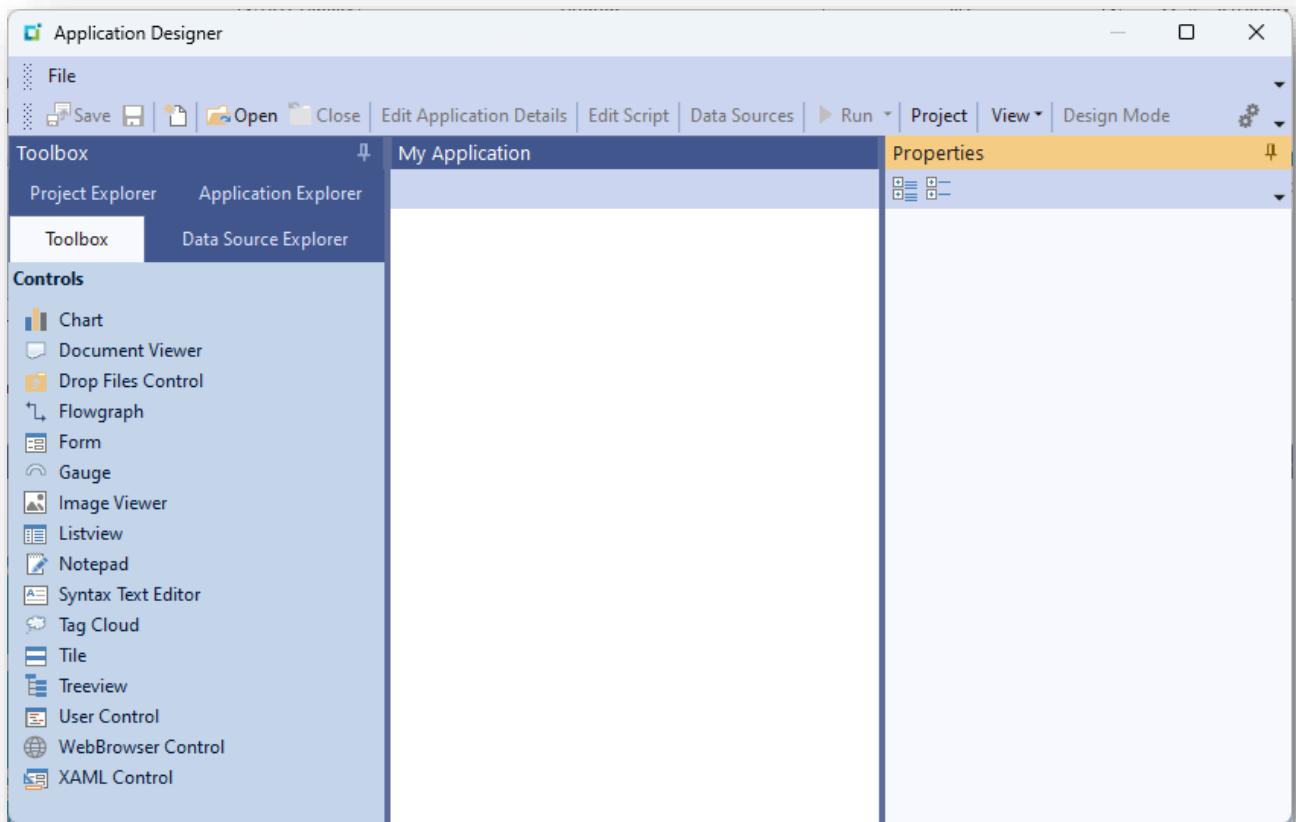
To design an application, you call **IMPIDE** from SYSPRO's main menu, or click on **Application Designer > Design Applications** in the **Administration** tab of the Ribbon Bar.



Note: You can also call IMPIDE passing it the application name to go straight into design mode, as in this example:



The initial view of the Application Designer looks like this:



TOOLBOX

The **Toolbox** tab shows the list of controls that can be added to an application. Up to 30 controls can reside in a single application.

Controls are arranged in your application within a docking pane layout, meaning that each control is contained in a docking pane which in turn is contained in a layout.

You can arrange these docking panes in any way you like within the application area simply by clicking on the docking pane header and dragging it to a new position, or by grouping them together in a TAB sequence. However, you arrange your panes is exactly how it will look when you run the application.

If you run an application whose option **In Development** is switched off, then any saved docking layout will be applied as the application loads. Generally, this all works okay, but if the application contains either new or deleted panes then it's possible that the application will not work correctly as the saved docking layout will not include these changed panes.

The Application Designer runtime will detect if the number of panes in the saved layout do not match that in the Application itself and will then automatically reset the docking layout.

Note: This automatic Reset Layout will not occur if there are any customized panes detected in the saved docking pane layout.

APPLICATION EXPLORER

The **Application Explorer** tab shows all the controls and toolbar buttons that have been added to an application. This is used to easily navigate to any control or toolbar button with a single click.

PROJECT EXPLORER

The **Project Explorer** tab shows all the projects and their resources. Click on a project to edit it or click on an application to open it for editing.

DATA SOURCE EXPLORER

The **Data Source Explorer** tab shows where a data source will be executed in the application.

Click on the hyperlinked execution point and this will navigate to the form field, the toolbar button or listview.

Click on the Data source name hyperlink to edit the data source.

MY APPLICATION

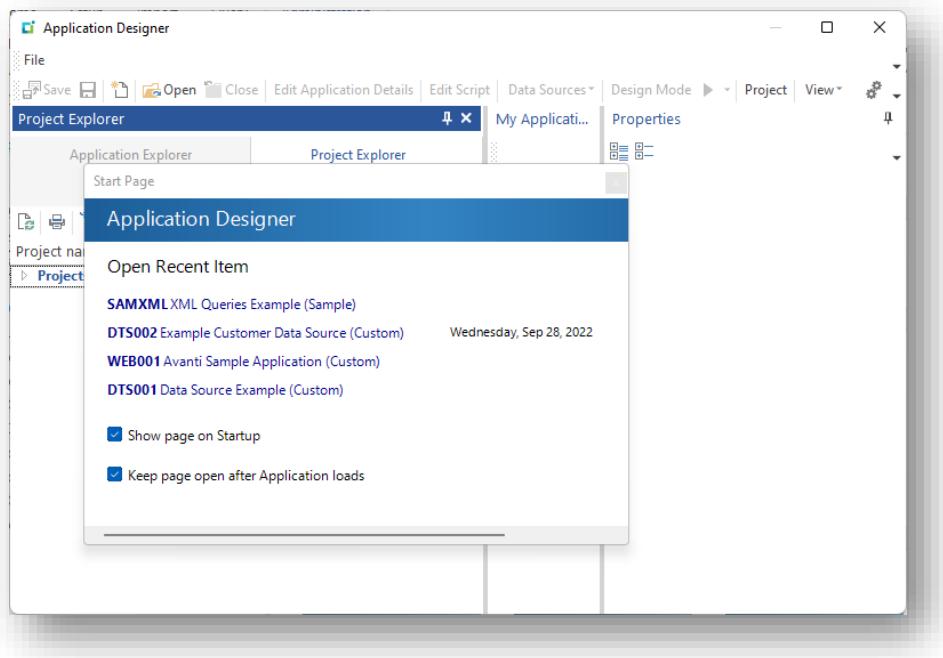
The area marked as **My Application** is where the application being designed will reside.

PROPERTIES

The **Properties** window will reflect the properties of the current control being edited or the main application details.

START PAGE

The Start Page window shows your most recent applications that have been opened:



With a single click on any item, you can view any application and begin to edit it.

Optionally, you can also define if the Start Page should automatically be displayed as the Application Designer loads and if it should remain open after editing an application.

DESIGNING FOR AVANTI

The Application Designer can only be run from within a SYSPRO Windows client. To create an Avanti web view for an application, proceed as follows:

1. Create the application using the **Application Designer** from within SYSPRO.
2. Create web views for the primary layout and all modal windows of the application from within the Designer.
3. Optionally, select the option to embed the web view(s) in the application.

See the section [Creating Avanti Web Views](#) for more information.

Note: The following controls are currently not supported in an Avanti web view:

- Flowgraph
- Tile Control
- XAML control

CREATING A NEW APPLICATION

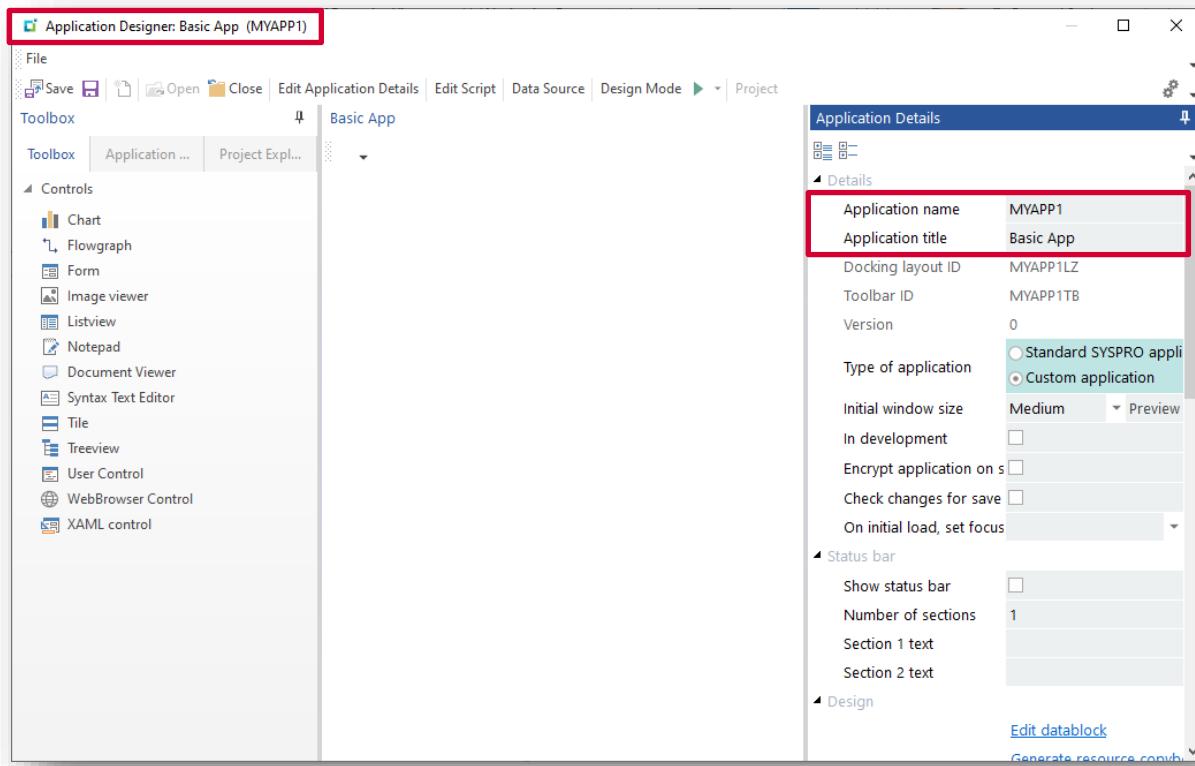
Click the **New** button to start creating a new application.

Enter your application name (the example is **MYAPP1**).

Considerations:

- This must be exactly 6 characters long and cannot contain special characters.
- This is the name of your application when you run it from a SYSPRO menu or in Avanti.
- Once the application has been saved, you cannot subsequently change the name.
- For a Custom application, the Application name must not begin with any of the standard SYSPRO module names (such as IMP, ARS, APS, INV etc.). This is to prevent any standard SYSPRO application from being overwritten.

Now enter the title of the application, which will appear as the window title:

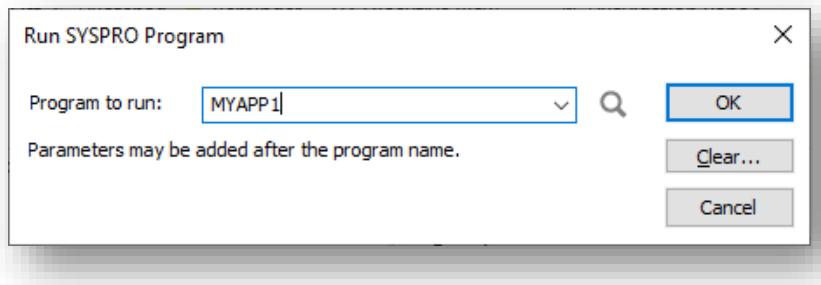


Note: It is important therefore that a naming convention is established so that if applications are being written by multiple developers, there is no clash in application names. This is particularly important when using the project export and import function, as it would be very easy to overwrite an application written by someone else.

TESTING THE APPLICATION

There are two ways to test the application:

1. Select **Save** to save the application, and then exit out of the Application Designer.
You can now run your application by pressing **CTRL+R** in the main SYSPRO menu and entering your program to run:



2. Alternatively, press **F5** to run the application.

This will automatically save the application and then run it in a separate instance of SYSPRO, the advantage being that you don't leave the application designer.

Your application will look like this:



Click **X** to close the application.

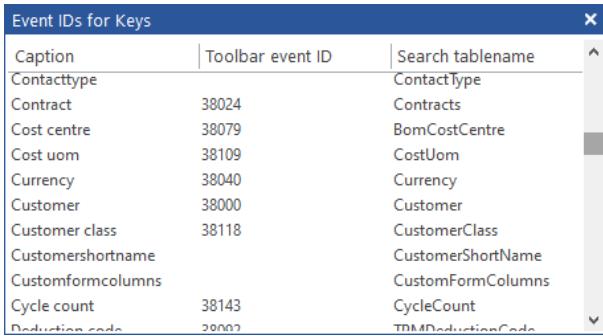
You are now ready to start changing some detail about the application, as well as adding controls to the application.

CHANGING APPLICATION DETAILS

In the Application Designer, click on **Edit Application Details** to change the properties of the application.

The following provides details for each option available:

Property	Explanation
DETAILS	
Application title	This indicates the title of the application that will appear in the window heading and be used to identify the application in the Recent Programs list in SYSPRO.
Docking Layout ID Toolbar ID	Each control and its associated (optional) toolbar have unique IDs. You don't generally need to know these IDs, but they can be referenced in script code, which is why they are generated by the designer, and you cannot change the values.
Version	This number increments every time the application is saved, unless the In Development option is enabled.
Type of application	<p>There are two types of applications:</p> <ul style="list-style-type: none"> ▪ Standard applications supplied by SYSPRO ▪ Custom applications <p>Note: Only the Custom option will be available outside of SYSPRO's development department.</p> <p>Custom applications are saved as TEXT files in the ...base\settings\AppDesigner folder on the local machine and then copied to the same-named folder on the server (if in client/server mode).</p>
Initial window size	<p>Select the appropriate initial window size for your application. Remember this is just the default view for the user; the user can resize and position the application window as required.</p> <p>Click the Preview button to see how the application will look – this floats the application window with the selected size.</p> <p>Double-click the docking pane caption to close the preview.</p>
In development	<p>Select this option to indicate that the application is 'in development'. When running the application and while this option is set, the application window position and any changes to Forms, List views, Toolbars and docking layouts will not be saved when the application exits.</p> <p>This is useful while developing the application.</p> <p>In addition, the Version number will not be incremented when the application is saved.</p> <p>Note: Remember to switch this option off before the application is deployed.</p>
Encrypt application on save	<p>This indicates that the application will be saved as both a text and an encrypted file (with an extension of .snk).</p> <p>The encrypted version has the advantage of being deployable to a site and run as a normal application, but the text cannot be humanly read or opened in the Application Designer.</p> <p>This protects the IP of the application.</p>

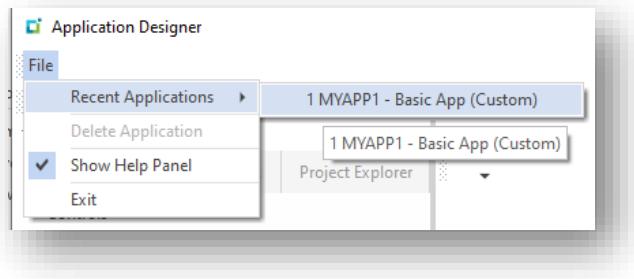
Property	Explanation																																							
Check changes for save	<p>This provides an automatic SAVE CHANGES mechanism.</p> <p>When clicking on the NEXT or PREVIOUS navigation buttons in the toolbar (and if the contents of any listviews, forms, notepads, or syntax editor have changed) then a 'Save Changes' task dialog will be shown.</p> <p>Event ToolbarSaveChanges is raised if the SAVE button is pressed Event ToolbarCancelChanges is raised if CANCEL button is pressed.</p>																																							
On initial load, set focus to	<p>Select the control to set focus to after the application loads.</p> <p>For an editable form, this means start editing on the first field on the form.</p> <p>For an editable listview, this means start editing on the first row in the listview.</p> <p>For a toolbar, this means setting focus to the edit control.</p>																																							
STATUS BAR																																								
Show status bar	Select this to show a status bar for the application.																																							
Number of sections	Select one or two sections.																																							
Section 1 text	<p>Enter text for the first section. You can dynamically change this text using the script callout ToolbarSetStatusBarText.</p> <p>This script statement will show the text Ready in section 1 of the status bar:</p> <pre>call CallSYSPROFunction(ToolbarSetStatusBarText, "DTSS001TB", "1 Ready")</pre>																																							
Section 2 text	Enter text for the second section. You can dynamically change this text using the script callout ToolbarSetStatusBarText .																																							
DESIGN																																								
Design toolbar	<p>Click to design the toolbar for the main application window.</p> <p><i>See the Designing a toolbar section for more information.</i></p>																																							
Event IDs for keys	<p>Click to see the event IDs and caption names for all key fields. Key fields are used for Predictive Search and Smart Links.</p> <p><i>For example: A caption of Customer class in a form or grid will enable predictive search for that field; similarly, a toolbar event ID of 38118 will enable predictive search in a toolbar.</i></p>  <table border="1"> <thead> <tr> <th colspan="3">Event IDs for Keys</th> </tr> <tr> <th>Caption</th> <th>Toolbar event ID</th> <th>Search tablename</th> </tr> </thead> <tbody> <tr> <td>Contacttype</td> <td></td> <td>ContactType</td> </tr> <tr> <td>Contract</td> <td>38024</td> <td>Contracts</td> </tr> <tr> <td>Cost centre</td> <td>38079</td> <td>BomCostCentre</td> </tr> <tr> <td>Cost uom</td> <td>38109</td> <td>CostUom</td> </tr> <tr> <td>Currency</td> <td>38040</td> <td>Currency</td> </tr> <tr> <td>Customer</td> <td>38000</td> <td>Customer</td> </tr> <tr> <td>Customer class</td> <td>38118</td> <td>CustomerClass</td> </tr> <tr> <td>Customershortname</td> <td></td> <td>CustomerShortName</td> </tr> <tr> <td>Customformcolumns</td> <td></td> <td>CustomFormColumns</td> </tr> <tr> <td>Cycle count</td> <td>38143</td> <td>CycleCount</td> </tr> <tr> <td>Deduction code</td> <td>38007</td> <td>TDMInductionCode</td> </tr> </tbody> </table>	Event IDs for Keys			Caption	Toolbar event ID	Search tablename	Contacttype		ContactType	Contract	38024	Contracts	Cost centre	38079	BomCostCentre	Cost uom	38109	CostUom	Currency	38040	Currency	Customer	38000	Customer	Customer class	38118	CustomerClass	Customershortname		CustomerShortName	Customformcolumns		CustomFormColumns	Cycle count	38143	CycleCount	Deduction code	38007	TDMInductionCode
Event IDs for Keys																																								
Caption	Toolbar event ID	Search tablename																																						
Contacttype		ContactType																																						
Contract	38024	Contracts																																						
Cost centre	38079	BomCostCentre																																						
Cost uom	38109	CostUom																																						
Currency	38040	Currency																																						
Customer	38000	Customer																																						
Customer class	38118	CustomerClass																																						
Customershortname		CustomerShortName																																						
Customformcolumns		CustomFormColumns																																						
Cycle count	38143	CycleCount																																						
Deduction code	38007	TDMInductionCode																																						

Property	Explanation
SCRIPTING	
Edit script	Click to edit the script associated with the application. See the Programming the Application section for more information.
Form OnGainFocus event required	This indicates that the script should be sent the GainFocus event for a form. If enabled: every time the user sets focus to any editable form field, then this event will be sent to the script. Note: This may impair performance of the application, so leave this option unchecked unless the application absolutely requires this event.
Toolbar OnGainFocus event required	This indicates that the script should be sent the GainFocus event for a toolbar entry field. If enabled: every time the user sets focus to any editable toolbar field, then this event will be sent to the script. Note: This may impair performance of the application, so leave this option unchecked unless the application absolutely requires this event.
AVANTI PROPERTIES	
Select web view to design	All applications that run in Avanti have a 'Primary Layout' web view (which is the main application's view) and a web view for each of the Dialog Boxes defined in the application. From the dropdown list, select the web view that you wish to design, then click Design Web View .
Design web view	Use the Visual Designer to design the selected web view. <ul style="list-style-type: none"> ▪ SYSPRO applications: The web view templates will be located in the ...base\SAMPLES folder. ▪ Custom applications: The web view templates will be located in the ...base\settings\AppDesigner folder. See the Designing a Web View section for more information.
Define Modal Windows	If you need more sophisticated modal windows in Avanti, see the Creating additional modal windows for Avanti section for more information.
Embed web view in application	This indicates that the web views designed for this application are embedded within the application text file. This makes it easier to deploy the application. Note: If you do not select this option, web views designed for the application will need to be deployed separately.
Property	Explanation

Show as modal window in Avanti	This indicates that this application should be shown in a modal window when running as a web view in Avanti. This is the appropriate choice if the application window is small enough not to have to consume the entire web page, as a modal window in Avanti means the Home Page workspace does not need to be refreshed.
APPLICATION PROPERTIES	
These properties are shown in the About this Application window (selected from the Application's Gear icon).	
Window icon	The default icon shown in the Window Title bar is a blue rectangle. However, you may select any .ICO file to be shown in the title bar. This file will need to be available on the client machine (if using client/server) so use the SAMPLES folder to store your ICO file as this folder is self-healed to the client. Then use this syntax, as an example: samples\myIcon.ico If the ICON is not valid when running the application then the blue rectangle will be shown instead. Note that this ICON will be ignored in Avanti.
Author	Enter the author of the application.
Release version	Enter the release version of the application.
Show notes in About window	This indicates that the Notes will be shown in the 'About this Application' window.
Notes	These notes are available for you to annotate your application. They do not affect the running of the application.

ADDING CONTROLS TO AN APPLICATION

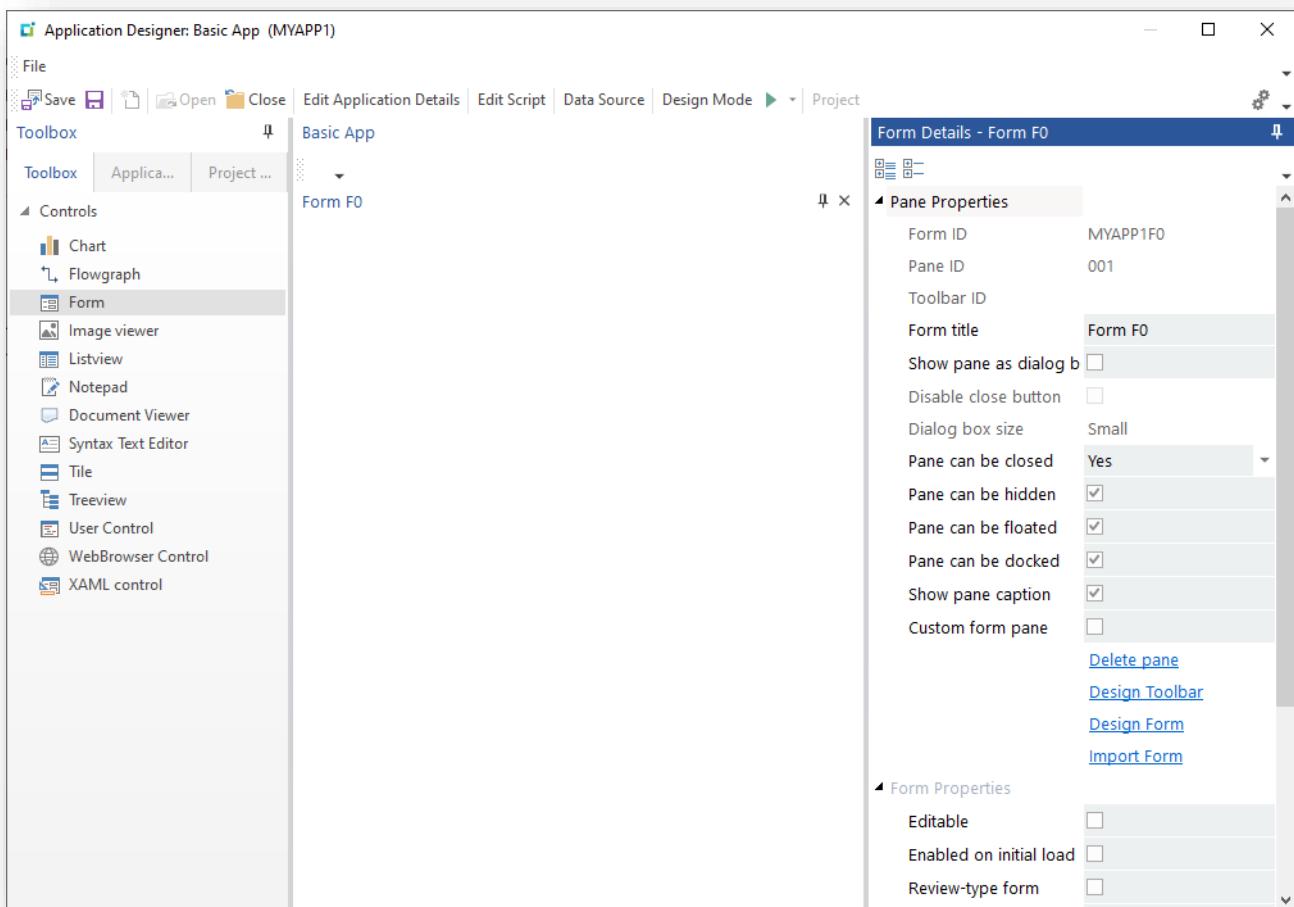
In the Application Designer, click on **File > Recent Applications** and select **MYAPP1**.



You will notice the option **Show Help Panel**. Just about every option in the designer has form field help. After a while, and when you've become familiar with the designer, you may want to switch off this option which will give you more space in the Properties forms.

To add a control to an application, click on the required Toolbar item. A control is always added to the top left of the application and from there you can position it anywhere as required. You can also make the pane 'closed' by default by clicking on the **X**, or 'hidden' by clicking on the **Pin** button. Remember that however you change a pane is how it will appear when a user first runs the application.

In the following example, a form has been added to the application:



All controls have a common set of Properties (i.e. Title, Pane properties, etc.) and then a set of specific properties.

The following explains the common properties:

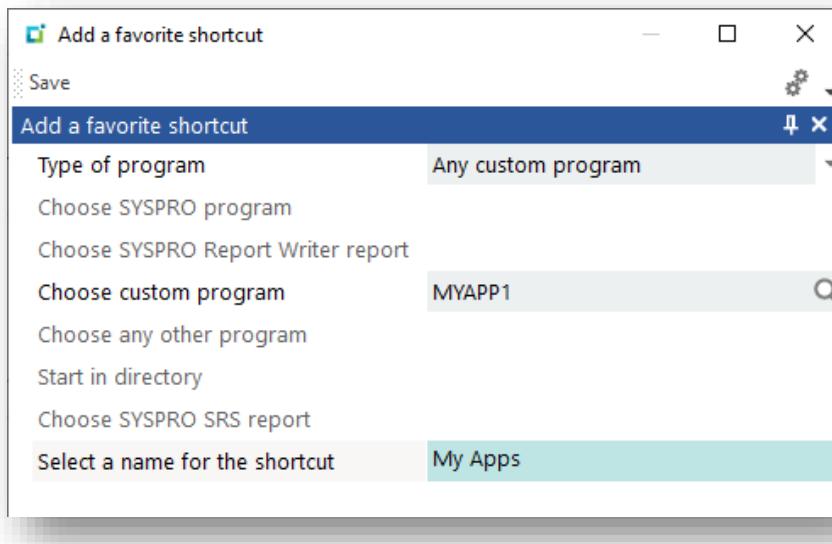
Property	Explanation
Title	The title is shown in the docking pane header and will be automatically translated.
Show pane as dialog box	This indicates that the pane will only be shown when the application invokes the ShowDialogBox script callout. Regardless of the appearance of the pane when designing, it will be automatically closed and set as 'disabled' as the application loads. A dialog box is a temporary window to retrieve user input or show some information. <i>See the section How to use a Dialog Box for more information.</i>
Property	Explanation

Show close button	<p>The options describe how the dialog box can be closed and whether the X is shown in the dialog box as a way of closing the pane.</p> <ul style="list-style-type: none"> Yes - show the close button (the X button). Clicking the close button will close the dialog box without intervention. No - the close button is not shown. The only way of closing the dialog box is by executing the CloseDialogBox callout in your script. Typically, you would add a button to the pane's toolbar that when clicked would execute the CloseDialogBox callout. Yes (with event) - show the close button. When clicked event DialogBoxClosing (60042) is fired. You can then execute the CloseDialogBox callout in your script to close the dialog box. <p>Options No and Yes (with event) allow you to decide if the dialog box can be closed, perhaps after checking if some or other form value has been entered.</p>
Dialog box size	<p>Select the initial size of the window, according to the requirements of the control to be hosted in the pane. This is just the initial size; the user can resize and reposition the window as required when running the application.</p> <p><i>See the In development option for more information.</i></p>
Pane can be closed	<p>This indicates if the pane can be closed by the user by clicking the X (the default option).</p> <p>Select No if you don't want the user to close this pane, perhaps if the pane content is central to the use of the application.</p> <p>Select Always closable if the pane can be closed, even if Roles are in use.</p> <p>It is recommended that at least 1 docking pane cannot be closed, so that the user cannot remove all panes and be left with a blank application.</p>
Pane can be floated	This indicates the pane can be dragged and floated.
Pane can be docked	This indicates the pane can be docked to another position in the layout.
Show pane caption	This ensures the pane caption is to be shown. If unchecked, a user will not be able to reposition the docking pane.
Delete pane	Click this to delete the pane.
Design Toolbar	<p>This lets you design a toolbar for the pane.</p> <p><i>See the Designing a toolbar section for more info.</i></p>

RUNNING THE APPLICATION

You can add your application to a menu, or your Favourites Tile.

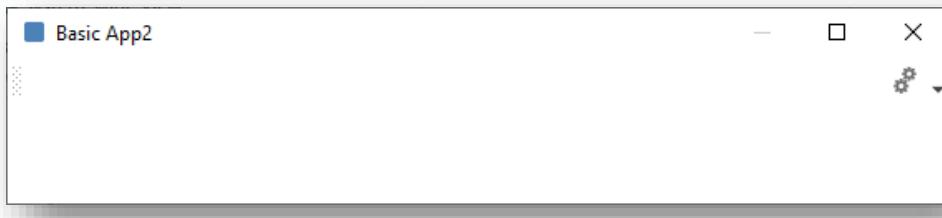
When adding a program select the **Custom program** option, as shown below:



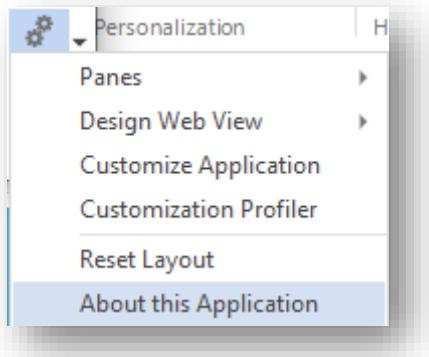
An application designed using the Application Designer works in a similar manner as any SYSPRO application, so much so that a user will not easily be able to distinguish between a standard SYSPRO and a custom application.

The following differences help a user identify a custom application:

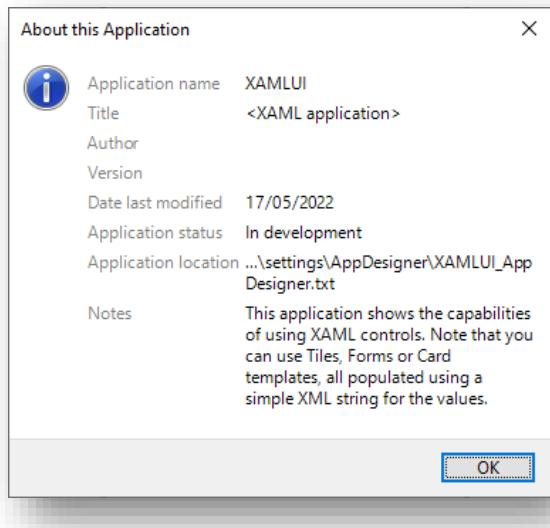
1. The Windows icon (a blue rectangle) is different to a SYSPRO icon. This helps identify this application as a 'custom' one.



2. There is an **About this Application** option available from the Gear:



This shows information relevant to the current application:



DESIGNING A TOOLBAR

The Toolbar control is probably one of the most important controls of the Application Designer. It is the key UI component that the end user engages with.

Toolbar controls reside at the top of the main application pane, but toolbars can be created at the top of any pane if required and multiple toolbars can be defined in a single pane.

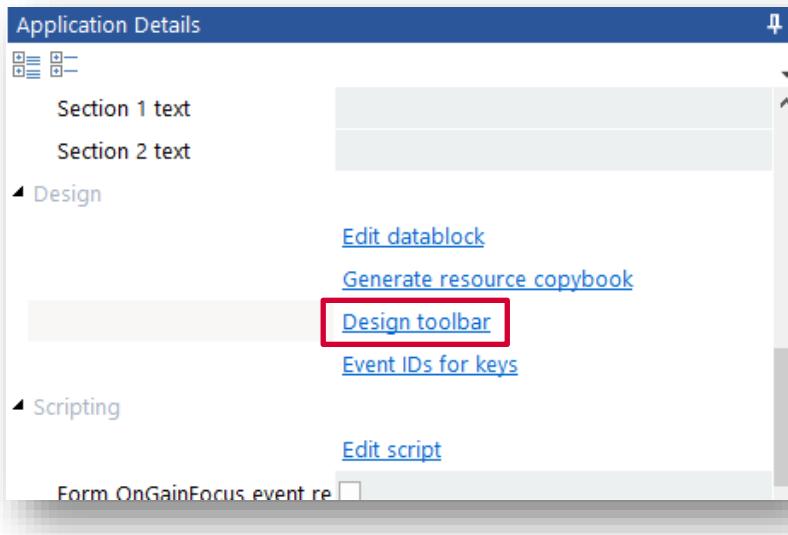
When a control is added to a toolbar, a unique Event ID is allocated to it. There are some special IDs that are reserved for KEY fields; if you have a caption of Customer, for example, then the event ID allocated will be 38000. The event ID will be allocated after typing in the toolbar caption.

When the control is used, an event is raised and the unique event id is passed back to the VBScript for processing along with relevant data, depending on the control type.

There are numerous Toolbar Callout functions that enable the VBScript to interact with the toolbar and its controls.

CREATING AND EDITING TOOLBARS

To design a Toolbar, click the **Design toolbar** hyperlink:



To Edit a toolbar and change properties of buttons, simply select the control from the toolbar and its properties will appear in the properties pane for editing.

To re-organize the sequence of controls on the toolbar or move buttons to sub menus (i.e. Popup, Button popup, and Split popup buttons) click **Customize**. Move the Customize window away from the toolbar so it can be seen and then click and drag any control to a new position.

GENERAL TOOLBAR FEATURES

Each control added to a toolbar is configured through the properties pane.

Here's an example of entering a **Customer** button:

Design Toolbar: Main Application Window

Control to add: Button Delete All Customize Import ...

Appearance

Caption	&Customer
Type	ComboBox
Event ID	38000
Associated with Predictive Search	Customer
Event IDs for keys	
Icon ID	0
Tooltip	Enter Customer
Style	Caption only
Bold	<input type="checkbox"/>
Width	150
Begin group	<input type="checkbox"/>
Close item on click	<input checked="" type="checkbox"/>
ComboBox entry allowed	<input checked="" type="checkbox"/>
Enter list	
Shortcut key	
Show dialog box	

Data sources

Data Sources to execute	Data Sources to Execute Customer Header, Customer Totals
-------------------------	---

Additional properties

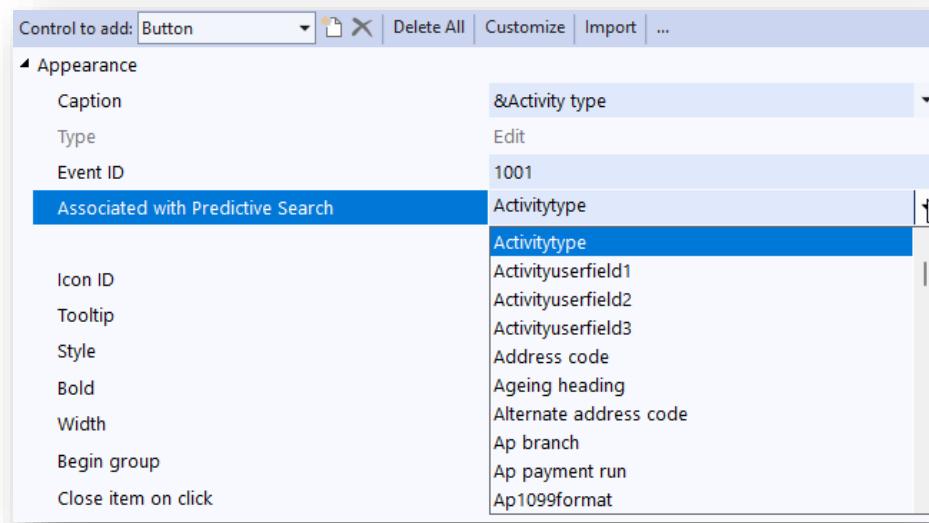
Description	datasource=Customer Header, Customer Totals;
Datablock name	Not set
Checked status	Not set
Enabled status	Not set
Hidden status	Not set
Visible status	Not set
Auto toggle check box	<input type="checkbox"/>
Auto add	<input type="checkbox"/>
Auto save	<input type="checkbox"/>
Auto complete	<input type="checkbox"/>
Validate KEY	Not set
Linked to toolbar control	
Program to run	
Edit hint	
Reset version	

The HELP panel in the Application Designer explains what each option means. Associating a control with a data source is explained in the section [Using Data Sources](#).

When typing in a caption, the Designer will automatically create a unique accelerator key. In the example below, an Ampersand (&) has been inserted before the first character.

The caption and tooltip for a toolbar can be forced to adhere to standards – please read the section **Validating word styling**.

Note: The entered caption is used to determine if there's a matching Predictive Search entry. If there isn't one then the property **Associated with Predictive Search** will be enabled and you can then select an appropriate search entry to be applied.



When the Application runs, pressing the Alt key and the character together will have the same effect as selecting it with a mouse. The IDE will ensure that the ampersand (&) character is inserted before the first character that is not duplicated.

When creating a combo box (or Edit control) and a Caption is recognized as a SYSPRO key field name; then the Application Designer will allocate a standard event ID to it, which in turn links the standard SYSPRO browse icon to the combo box (and with it, all the functionality and search capability of the standard product).

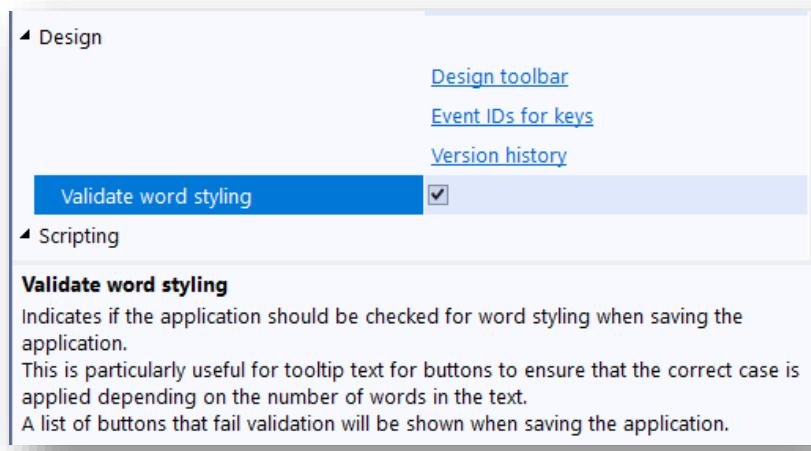
When a control raises an event, the following data is passed to the VBScript:

- **EventType** is of type 'ToolbarEvent'
- **EventName** is the name of the toolbar
- **EventID** is the unique ID allocated to the control
- **Param1,2,3** holds data depending on the control type

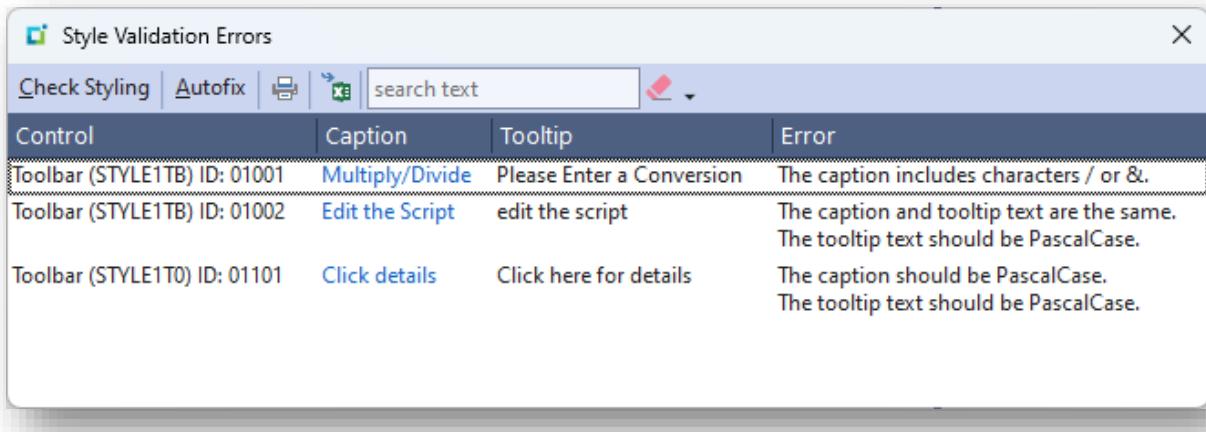
VALIDATING WORD STYLING

When designing an application, you can validate and auto fix toolbar button captions and tooltip text. Toolbar captions should generally be in PascalCase, as should the tooltip text if it contains 5 or less words, and it can be time-consuming to ensure all toolbar captions and tooltip text conform to these standards.

To check for word styling, just check the option **Validate word styling** in the Application Details pane:



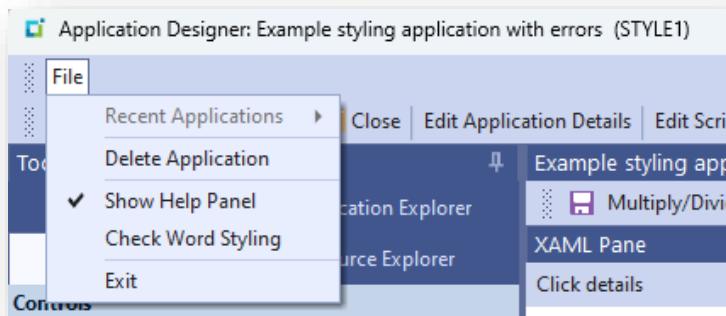
When you save the application, and if any toolbar controls fail the validation, then the following window will be shown, listing the errors:



Click on the hyperlinked caption to edit the toolbar button manually or click **Autofix** which will automatically fix most of the issues listed for the toolbar captions and tooltip text.

Click **Check styling** to check for any styling errors.

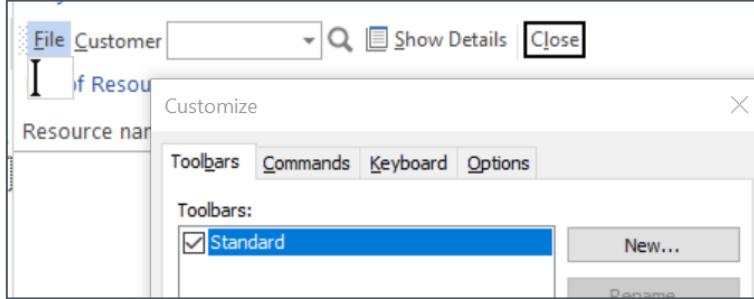
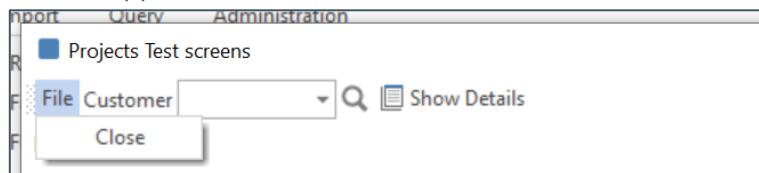
You can also validate your application at any time by clicking **Check Word Styling**:



TYPES OF CONTROL

The following is a list of controls that can be added to a toolbar:

Note: When any control on the toolbar is executed, it raises a **ToolbarEvent** event type passing the **Event ID**.

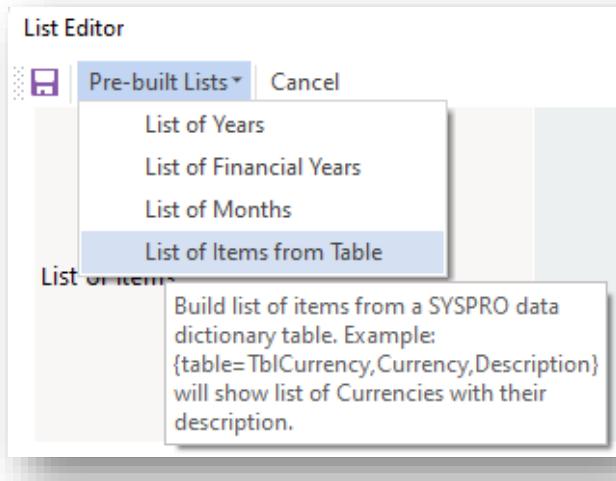
Property	Explanation
Button	<p>Traditional UI button.</p> <p>No values are included in Param1,2,3.</p>
Button popup and Popup	<p>Used as a “pull down menu”.</p> <p>This control displays other toolbar controls such as buttons. It does not create its own event if pressed, it is just a container for other buttons.</p> <p>A Popup control differs slightly from a Button Popup in that it shows an expand button:</p>  <p>To set this up:</p> <ol style="list-style-type: none"> 1. Create the popup. 2. On the toolbar, create each of the buttons that will reside in this popup. 3. Once the components have been created, use the toolbar customization screen (see above) to drag the child buttons into the popup button:  <p>4. Go to customize toolbar and select child button, then move to the popup menu.</p>  <p>And when the App runs:</p> 
Split button popup	<p>This is a combination of a button and popup menu.</p> <p>No values are included in Param1,2,3 when the button is pressed.</p> <p>If the down arrow is pressed, a sub menu of other controls is displayed.</p> <p>The popup menu behaves exactly as the Button popup.</p>

Property	Explanation
Combo box	<p>This is used to hold a list of items.</p> <p>The list of items can be static (i.e. entered at design stage and never changed) or dynamic (i.e. populated with data either from the VBScript or directly from SYSPRO).</p> <p>When an option is selected:</p> <ul style="list-style-type: none"> ▪ Param1 holds the item selected ▪ Param2 holds the caption of the control <p><i>See also Drop down lists for a combo box control</i></p>
Edit	<p>This is a text box that enables data to be entered into the toolbar.</p> <p>Entering data and pressing Return or Enter raises the event:</p> <ul style="list-style-type: none"> ▪ Param1 holds the data ▪ Param2 holds the caption of the control
Checkbox	<p>This is a single checkbox that can be ticked or not.</p> <ul style="list-style-type: none"> ▪ Param1 holds the check status. <p>0 indicates unchecked 1 indicates checked</p>
Radio button	<p>This is a single radio button that can be checked or not.</p> <ul style="list-style-type: none"> ▪ Param1 holds the check status. <p>0 indicates unchecked 1 indicates checked</p>

DROP DOWN LISTS FOR A COMBO BOX CONTROL

Drop down lists can be manually entered lists or can be constructed using one of the **Pre-built Lists**. Pre-built lists can be a list of years or financial years, or even a list of items derived from a data dictionary table – this can save you from having to create code to build up a list of items and populate a dropdown list.

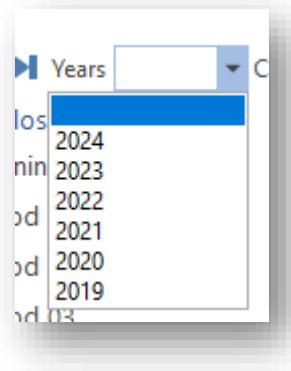
Click on **Pre-built Lists** and select one of the available menu options:



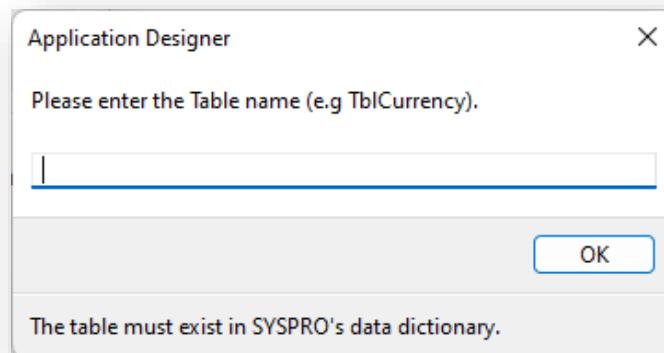
Selecting **List of Financial Years**, for example, will insert this syntax into the list editor:

```
{blank, financialyear-5}
```

This is then rendered as follows when the application runs:

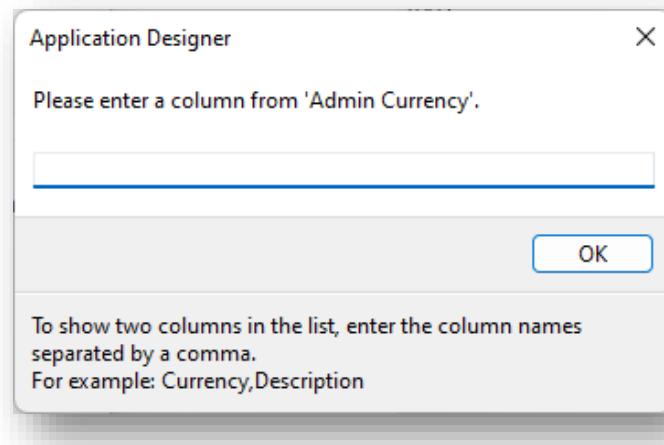


You can also construct a list from any SYSPRO data dictionary table. For example, suppose you wish to show a list of currencies in the toolbar, then select the option **List of Items from Table** and this prompt will appear:



Enter **TblCurrency** and click **OK**.

You will be prompted to enter a column name that exists in the table:



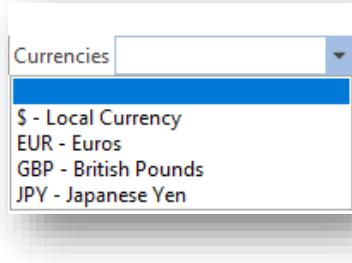
Note: You can enter up to two column names separated by a comma.

Enter **Currency, Description** and click **OK**.

The following syntax will be inserted into the editor:

```
{table=TblCurrency,Currency,Description}
```

Save the application with this changed list and run the application. It will render as follows:



Note: A maximum of 100 items will be returned from the table to be displayed in the drop-down list.

USING KEY EVENT IDs FOR TOOLBAR BUTTONS

If you entered a caption which is associated with a KEY field (e.g. Customer, Stock code etc.) the following useful features are available:

- **Automatic Browse Functionality**

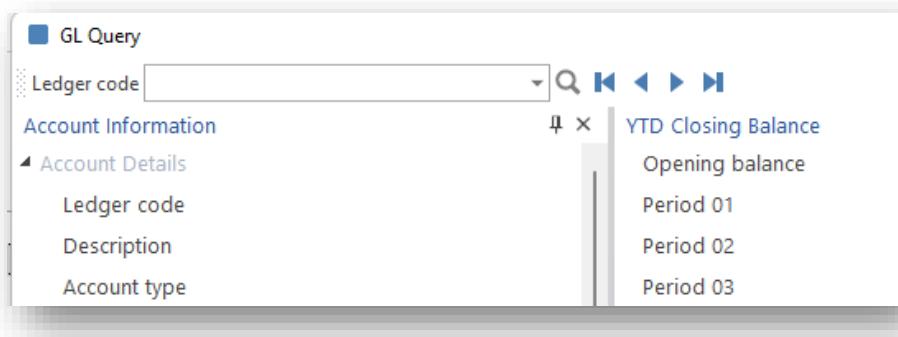
If the toolbar button is editable, then a **BROWSE** button is automatically added to the toolbar. This allows for built-in browsing capability when running the application.

- **Navigation and Multimedia functions**

You can add navigation buttons (i.e. First, Last, Next and Previous) and Multimedia buttons (i.e. Play and Edit) to a toolbar with automatic execution.

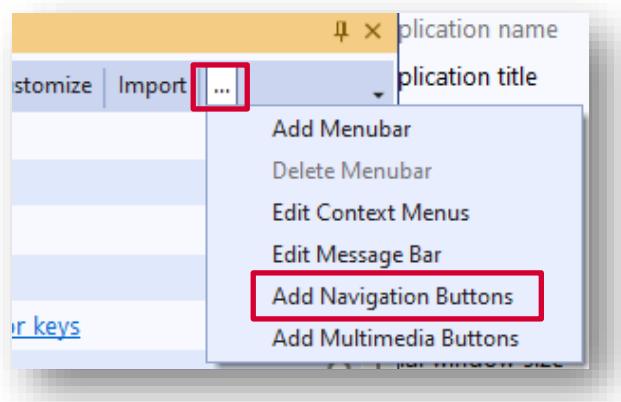
Navigation functions...

Here's an example of a toolbar with a toolbar control labelled **LEDGER CODE** and the four navigation buttons:

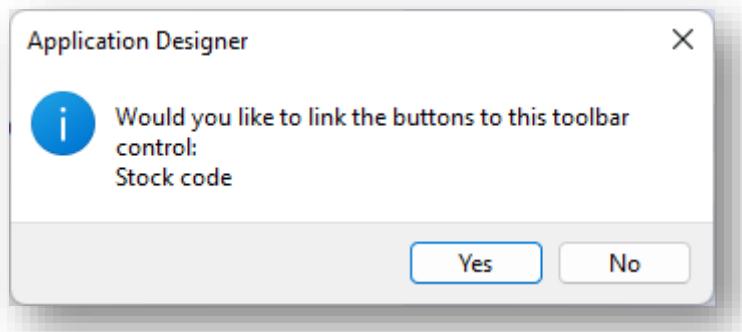


When clicking on **First**, **Next**, **Previous** and **Last** navigation buttons; the App Designer will automatically extract the value of the linked toolbar control to retrieve the Next and Previous KEY values.

If you haven't yet added navigation buttons to the application's toolbar, you can do this by selecting to design the application's toolbar, then click on the ellipses (i.e. ...), followed by the **Add Navigation Buttons** option:



If you have already added an editable toolbar control to your application, then this message will appear:



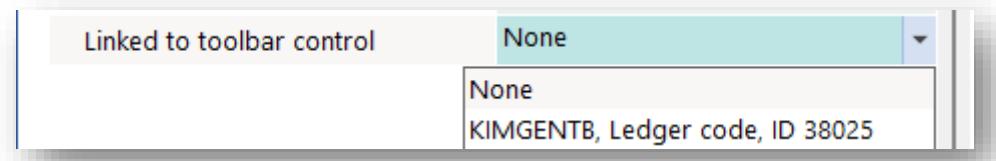
If you select **YES**:

The navigation buttons will be added to the toolbar and automatically linked to the indicated toolbar control.

If you select **NO**:

The navigation buttons will still be added to the toolbar, and subsequently you can define the linked toolbar control property.

You can manually update the link for any navigation button by selecting the toolbar control in the property **Linked to toolbar control**:

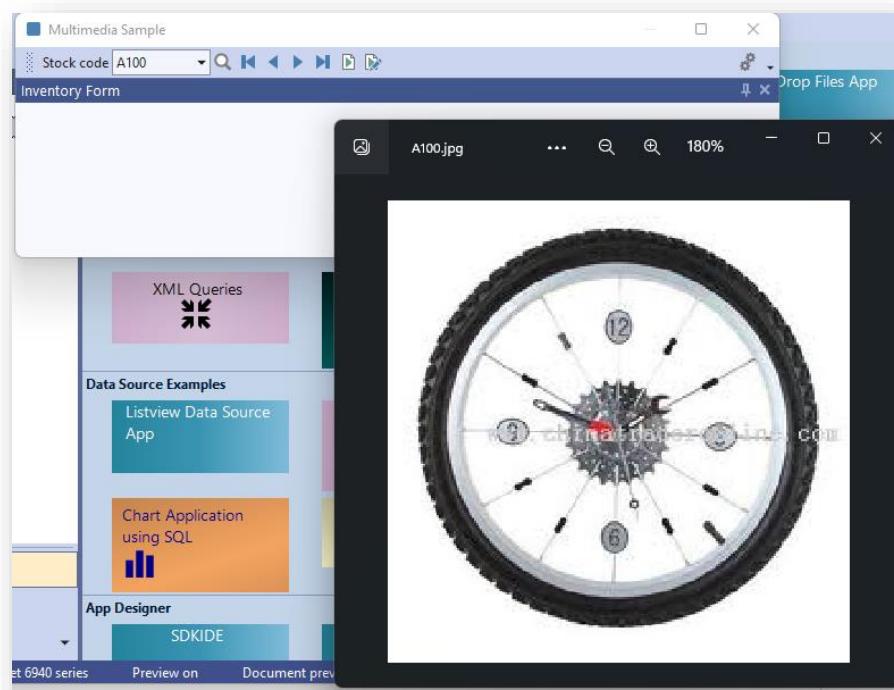


Multimedia functions...

Similarly, you can add **Multimedia buttons** to the toolbar.

If you haven't yet added Multimedia buttons to the application's toolbar, you can do this by selecting to design the application's toolbar, then click on **Add Multimedia Buttons** (i.e. **Play** and **Edit**).

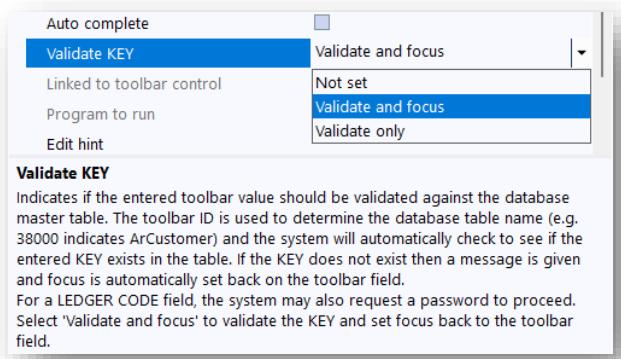
When selected, these buttons call up the **Multimedia** program (i.e. **IMPMCI**) to either show linked multimedia objects, or to allow the editing of such, as shown in the screen shot below:



▪ **Validating toolbar fields without code**

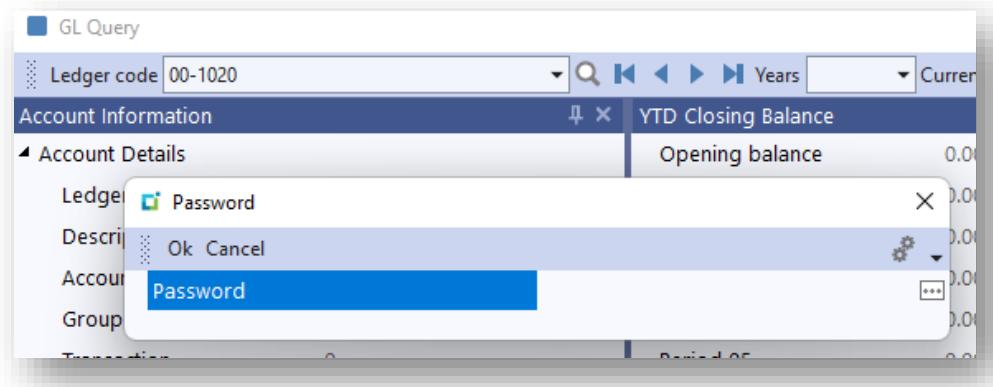
Often an application will have one or more KEY fields in an application's main toolbar that need validating to ensure that the KEY value entered is valid. You could do the validation using the application's VBScript, or you could link the toolbar field to a data source.

There is another way of achieving the validation, simply by using the property **Validate KEY** against an editable toolbar control:



This option will only work if the ID of the toolbar falls in the **38000** to **38250** range.
This range of IDs indicates the associated SYSPRO table name and will be used to validate that the entered KEY value exists in the table.

Additionally, if the toolbar ID is for a **LEDGER CODE**, then the password for the entered KEY will be automatically requested:



If the entered KEY is not valid:

- Focus is automatically set back on the toolbar field.
- No further processing will occur except to raise the **ValidateKeyFail** event (for EventType **DataSourceEvent**) in the script. This event allows you to clear forms, listviews or any other processing as required.

For example:

```
if EventType = DataSourceEvent then
    if EventID = ValidateKeyFail then
        call CallSYSPROFunction(FormClearData, "XMLIN1FO", " ")
        call CallSYSPROFunction(ListviewClearGrid, "XMLIN1L0", " ")
        call CallSYSPROFunction(NotepadClear, "XMLIN1N0", " ")
        call CallSYSPROFunction(DocumentShow, "XMLIN1P0", " ")
    end if
end if
```

If the entered KEY is valid:

- Data sources will be executed and the script **OnUserEvent** will be triggered.
- Focus will be set back on the toolbar field if the **Validate KEY** option is defined as **Validate and focus**.

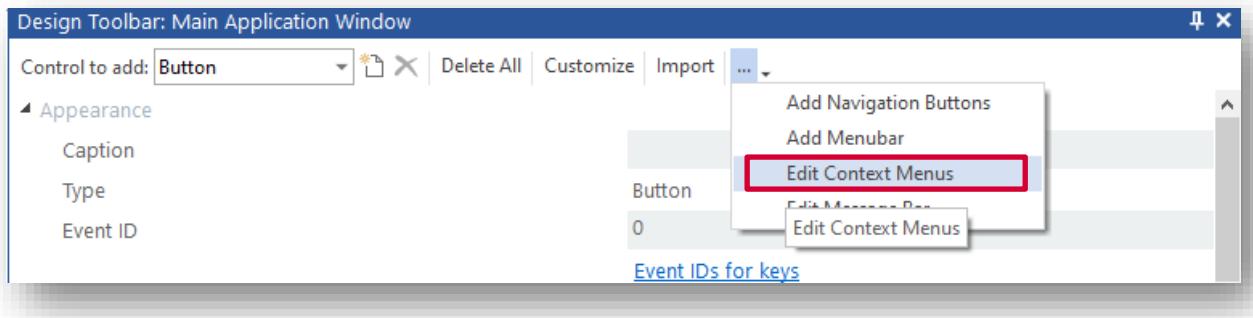
CALLOUT FUNCTIONS

The list below indicates the callout functions that allow the VBScript to engage with the toolbar:

Toolbar callouts		
Set focus to a toolbar entry field	(ToolbarSetFocus, "xxxxxxxx", "nnnnn")	The toolbar field must be editable.
Set checkmark	(ToolbarSetCheckMark, "xxxxxxxx", "nnnnn")	
Unset checkmark	(ToolbarUnsetCheckMark, "xxxxxxxx", "nnnnn")	
Get checkmark state	(ToolbarGetCheckMark, "xxxxxxxx", "nnnnn")	Returns 1 (checked) or 0 (unchecked).
Toggle checkmark state	(ToolbarToggleCheckMark, "xxxxxxxx", "nnnnn")	Toggles the checkmark state (if checked, then set unchecked; if unchecked, then set checked).
Get value of a toolbar field	(ToolbarGetValue, "xxxxxxxx", "nnnnn")	Returns the value of the toolbar edit control. If it is a combo box control with enumerators, then the enumerator is returned.
Set value of a toolbar field	(ToolbarSetValue, "xxxxxxxx", "nnnnn new value")	The toolbar field must be an edit control.
Set combo box index	(ToolbarSetIndexPointer, "xxxxxxxx", "nnnnn index pointer")	This sets a specific list item by index number. The toolbar field must be a combo box control.
Set combo box index by value	(ToolbarSetIndexByValue, "xxxxxxxx", "nnnnn enumerator")	This sets a specific list item using an enumerator. The toolbar field must be a combo box control.
Set tooltip of a toolbar field	(ToolbarSetTooltip, "xxxxxxxx", "nnnnn tooltip")	
Change the status bar text	(ToolbarSetStatusBarText, "xxxxxxxx", "1 text")	Requires the section number (1 or 2) and the new text.
Disable a button	(ToolbarDisableButton, "xxxxxxxx", "nnnnn")	
Enable a button	(ToolbarEnableButton, "xxxxxxxx", "nnnnn")	
Add to end of dropdown list	(ToolbarAddToDropDown, "xxxxxxxx", "nnnnn item caption")	The toolbar field must be a combo box control.
Set a dropdown list	(ToolbarSetDropDownList, "xxxxxxxx", "nnnnn list")	The toolbar field must be a combo box control. The items in the list must be separated by a semi-colon.
Show a context menu	(ToolbarShowContextMenu, "xxxxxxxx", "nnnnn")	Returns the ID of the selected context menu item, or '00000' if no menu item selected.

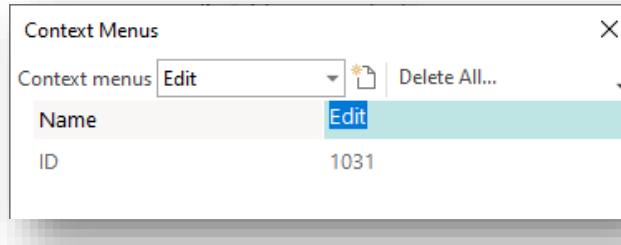
DESIGNING A CONTEXT MENU

A context menu is a pop-up menu that provides shortcuts for actions. A context menu is held within the main application toolbar:

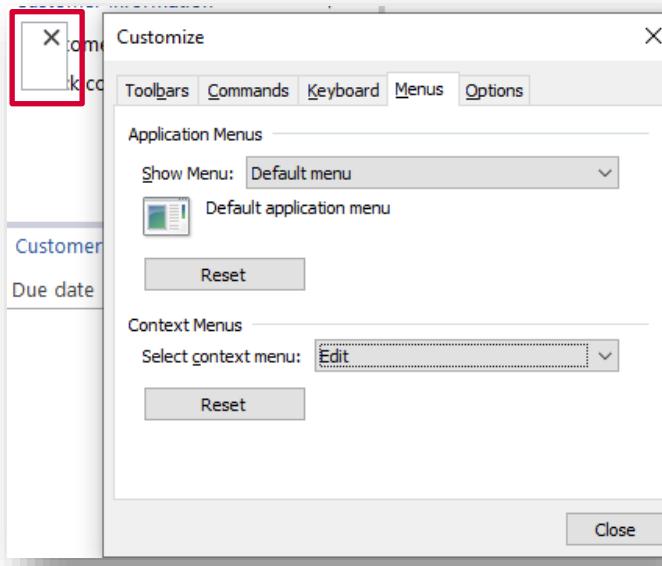


Add a context menu and define the **Name** for the menu.

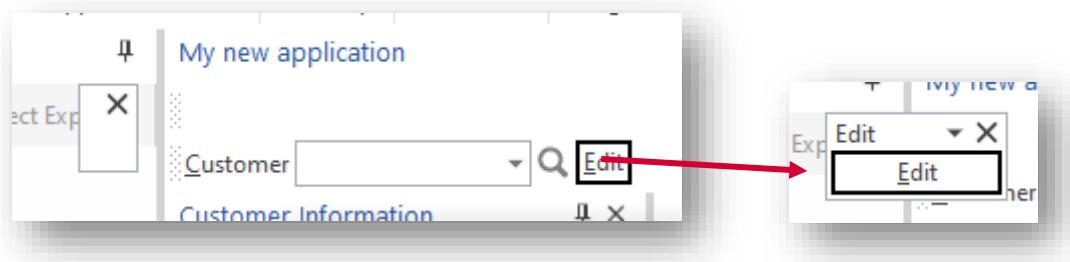
Note the **ID** (e.g. 1031) of the context menu, as you will reference that when showing the context menu.



To add items to this context menu, click on **Customize** and then select the **Menus** tab. Then select the required context menu. The context menu is then displayed:



You can now drag any items from the main toolbar onto the context menu:



Close the Customize dialog box.

To show the context menu in your script, double-click the **Show a context menu** callout and select the context menu you wish to show:

Set a dropdown list	"xxxxxxxx", "nnnnn item caption") (ToolbarSetDropDownList, "xxxxxxxx", "nnnnn list")	The toolbar field must be a combo box control. The items in the list must be separated by a semi-colon.
Show a context menu	Select Control Name	contextMenu, 'n')
Check for changes	APP001TB 01031 Edit	changes, "xxxxxxxx", Returns a status of controls that have changed, and shows a Task Dialog to save changes.

If an item is selected from the context menu, its control ID will be returned in **ReturnValue**.

Considerations:

- You must first define whatever buttons you want in your context menu on your main application toolbar. Once they are shown on the toolbar, they can then be dragged onto your context menu.
- This technique will work both in SYSPRO and in Avanti.
- You can show and edit context menus from the Application Explorer view.

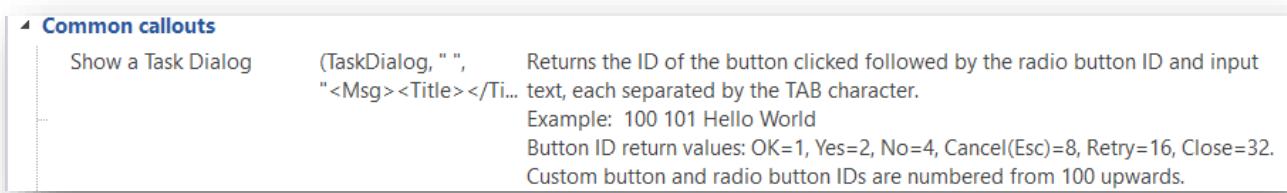
CREATING TASK DIALOGS

A task dialog is a dialog box that can be used to display information and receive simple input from the user.

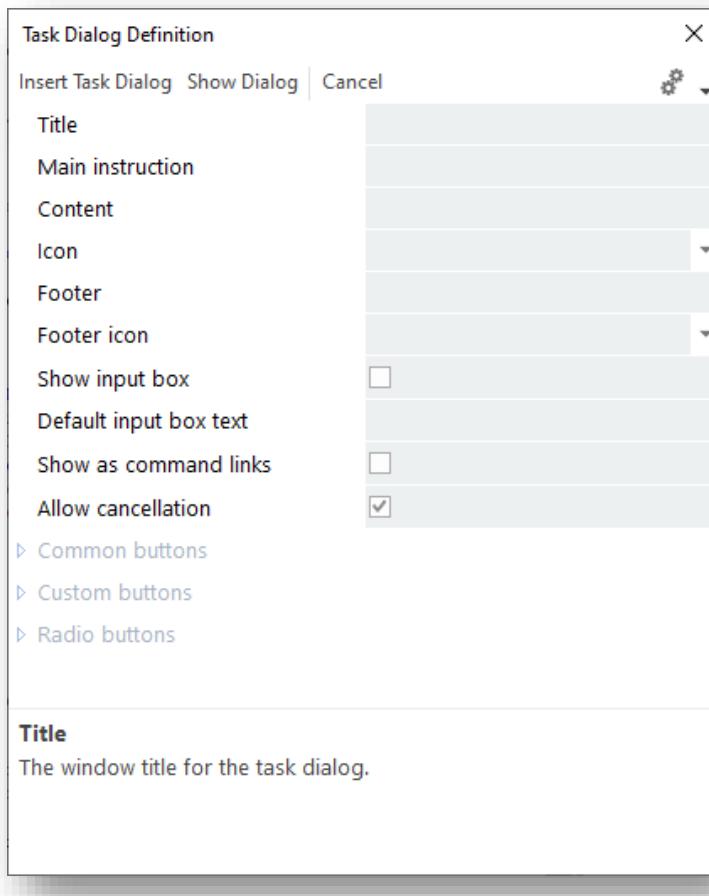
The **TaskDialog** Callout is the Application Designer's equivalent to VBScript's MSGBOX command but is far more flexible, enabling the developer to provide **toast** notifications, traditional message box messages with buttons, text, and radio button interaction with the user.

Its real flexibility comes when the different categories are mixed. For example, using standard and custom buttons with an input dialog. Or using standard and custom buttons with radio button dialog.

The **TaskDialog** callout is found in the **Common callouts** section:



Double clicking **Show a task Dialog** will launch the Task Dialog Definition window, which helps create the VBScript to display a task dialog.



Click **Insert Task Dialog** to insert the Task Dialog code at the current script position.

Show Dialog shows a preview of the dialog.

Pressing any **Common buttons**, **Custom buttons** or **Radio buttons** will display the return values that will be sent to the VBScript. This enables testing of the dialog without having to run code.

The Task Dialog control will always return 3 variables delimited by a tab character:

- The Button ID
- The radio button ID
- The input text

If the variable isn't appropriate for the dialog, then "" is returned.

The return value can then be split into an array and each individual value can be accessed.

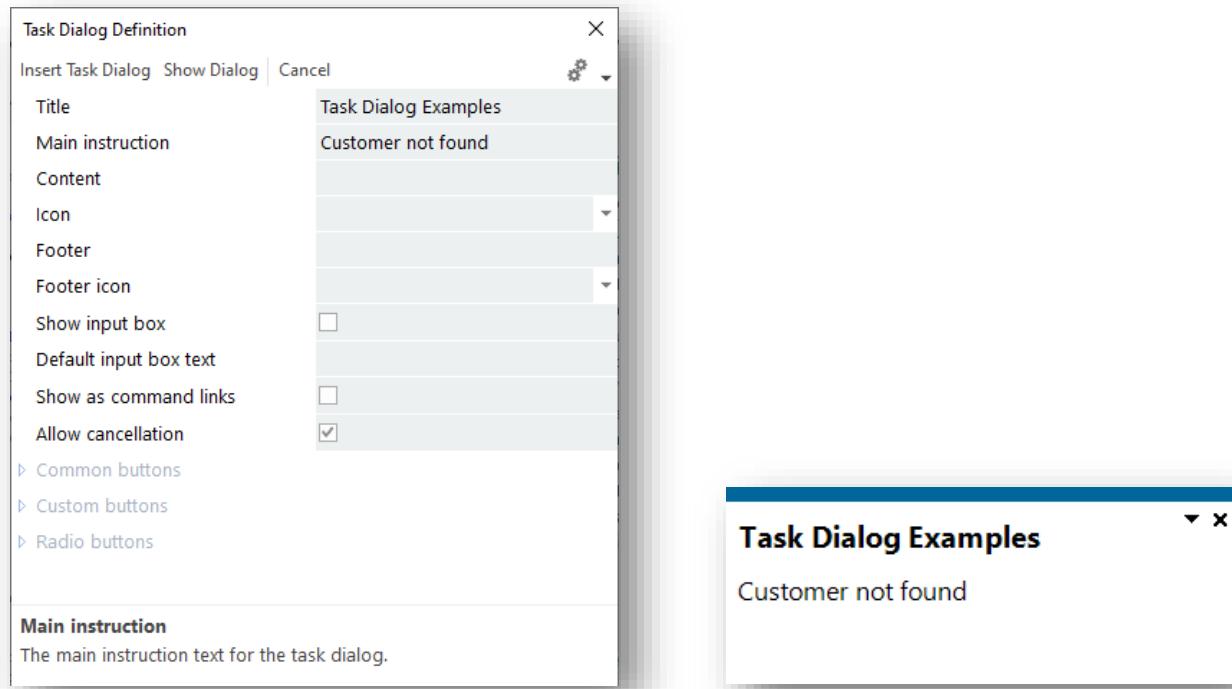
For example:

```
144 Function DisplayReturnValues(strRetVal)
145     dim i
146     dim returnValArray
147
148     returnValArray = split(strRetVal,vbTab)
149
150     'for i = 0 to ubound(returnValArray)
151     '    msgbox "i = " & i & " " & returnValArray(i)
152     'next
153
154     msgbox "Button ID: '" & returnValArray(0) & "' & vbcrlf &
155         "Radio Button ID: '" & returnValArray(1) & "' & vbcrlf &
156         "Input Text: '" & returnValArray(2) & "'", , "Task Dialog return values"
157
158
159
160 end function
```

TOAST NOTIFICATIONS

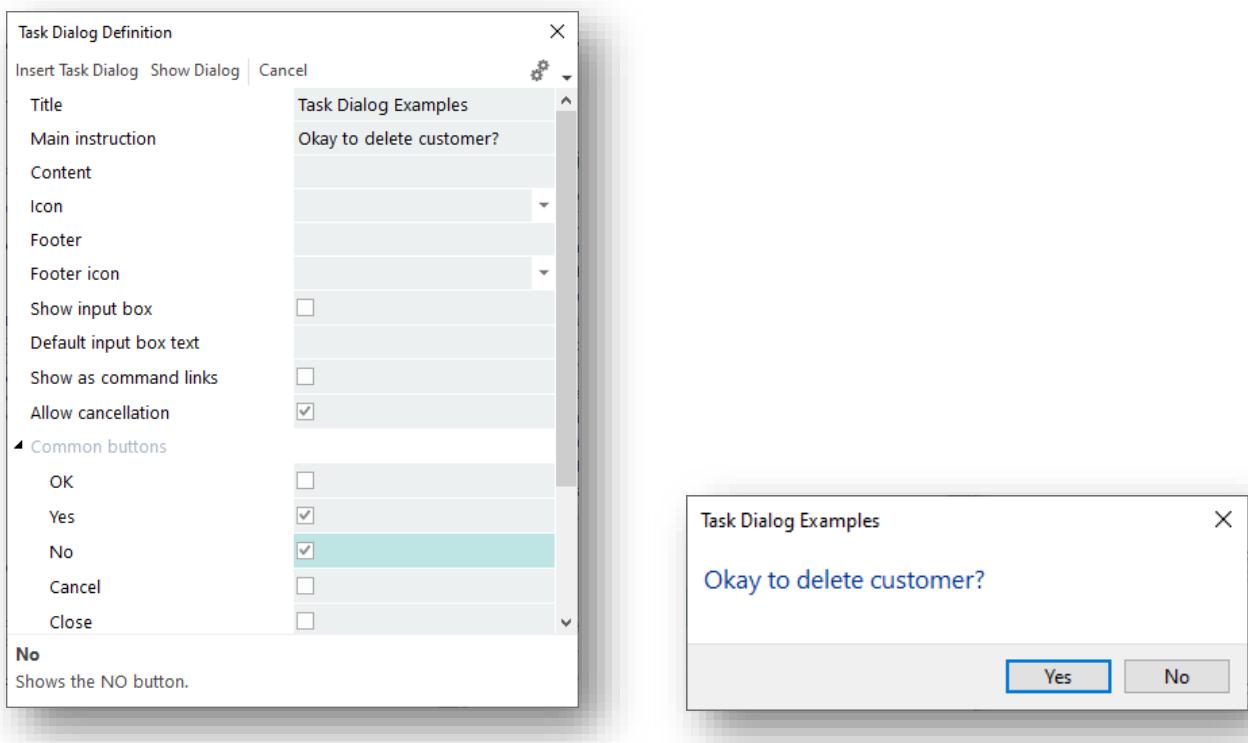
Toast Notifications are popup dialogs that appear on the screen for a few seconds before disappearing automatically. They do not require any operator interaction and are useful to guide the operator and inform them of progress.

Defining a Task Dialog without any buttons or input box will automatically show a Toast Notification:



COMMON BUTTON TASK DIALOG

This form of the Task Dialog mimics the VBScript MSGBOX command:

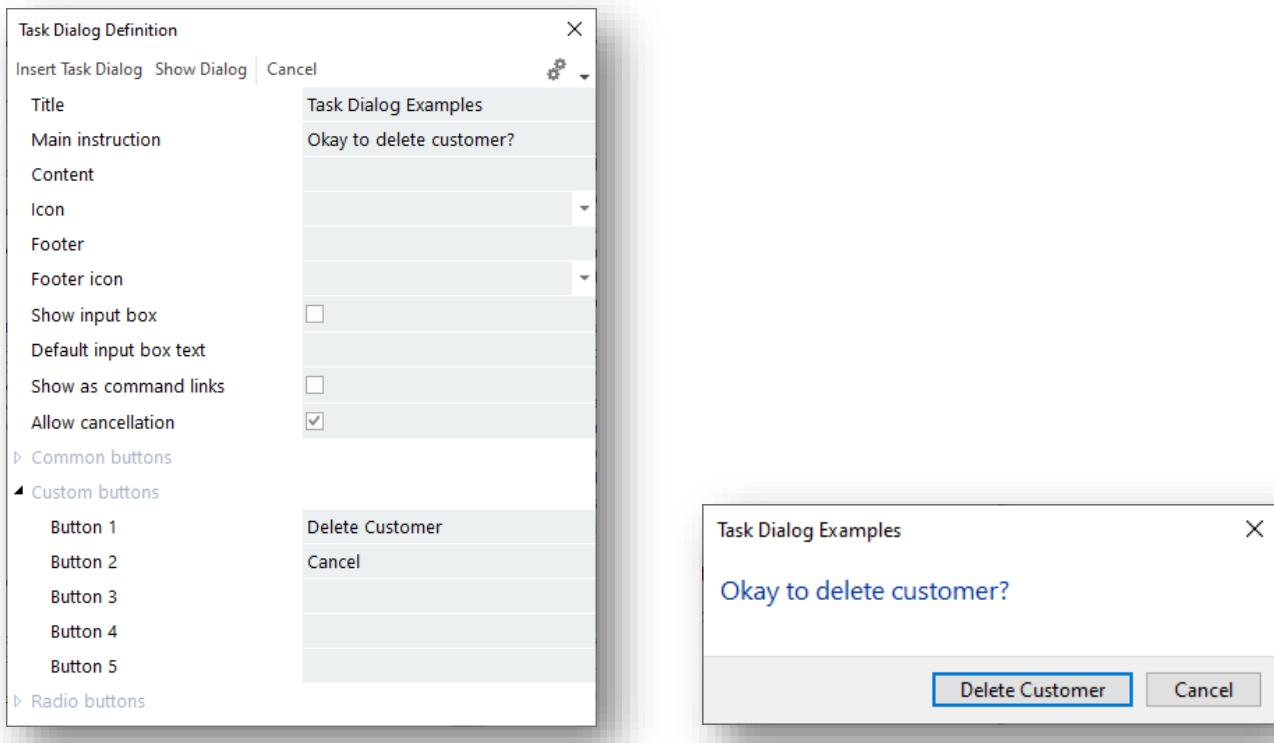


In this example, selecting **Yes** gives a return value of **002**, whereas selecting **No** returns **004**.

Note: The **Default button** property applies to the custom buttons as well.

CUSTOM BUTTON TASK DIALOG

The custom button option allows the developer to set the caption of the buttons.

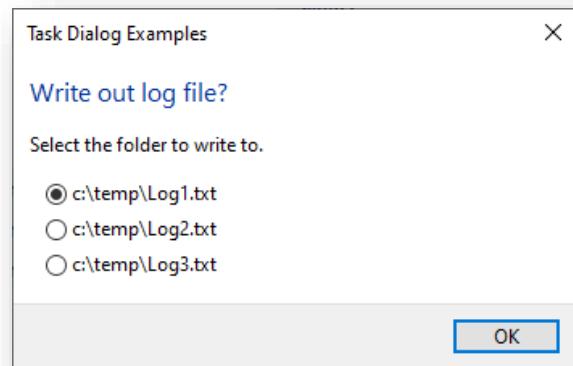
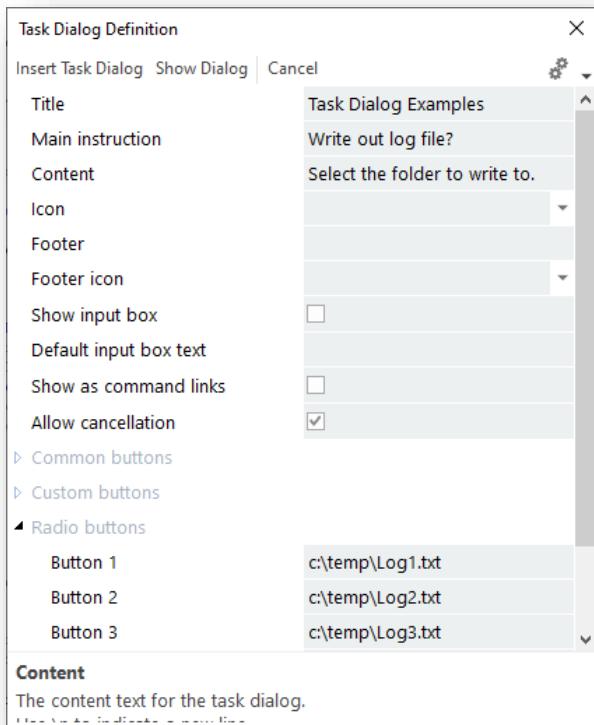


In this example, **Delete Customer** returns **100**. **Cancel** returns **101**.

If you need to set a default button, then use the **Default button** property in the common buttons section.

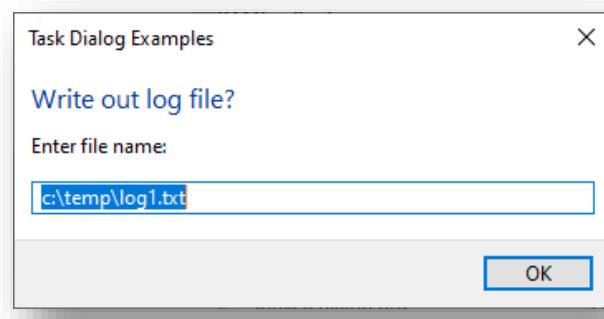
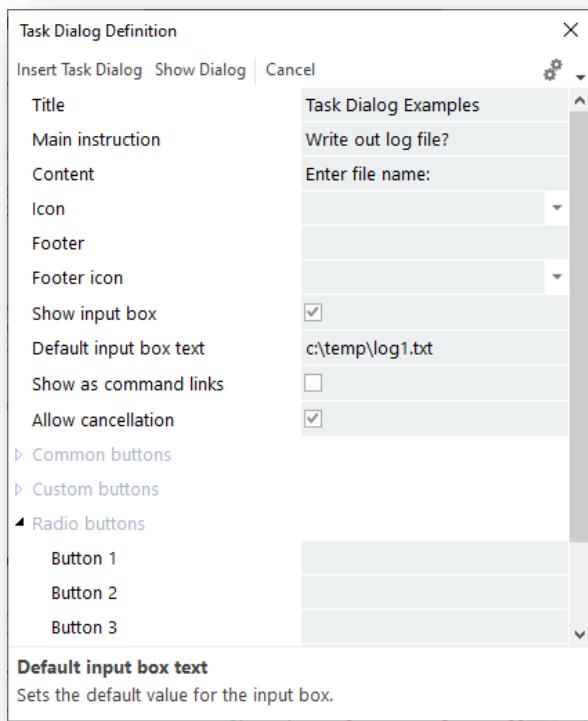
RADIO BUTTONS

The radio button option provides a mechanism to mix buttons (both common and standard) with a radio button option:



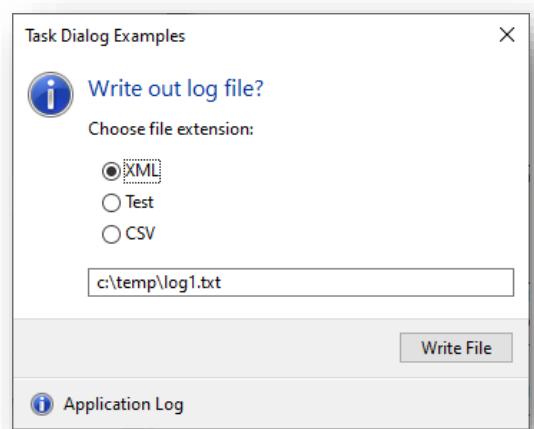
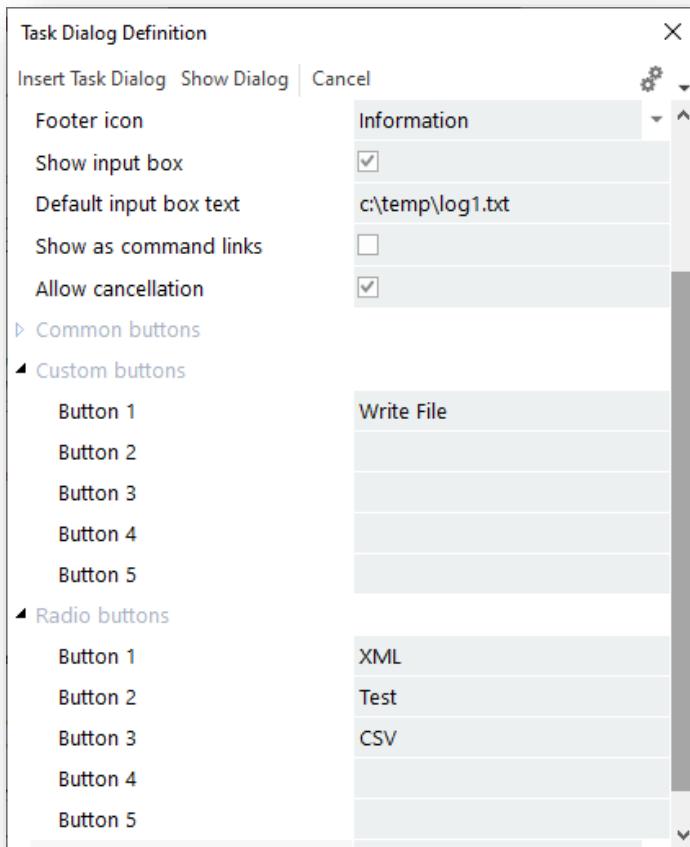
INPUT BOX DIALOG

The input box dialog enables entry of text into the dialog:



MIXING IT UP A BIT

It is possible to use properties from each of the sections and build a task dialog that is a combination of buttons, radio buttons and input text:



CREATING AVANTI WEB VIEWS

It is not possible to create an Avanti web view directly without having first created the application in the Application Designer in SYSPRO. While the content in SYSPRO and in Avanti is generally the same, the layout of a web view can be and is often different from a Windows Client User Interface.

For example:

Toolbar buttons are normally left-aligned when in SYSPRO, whereas in Avanti toolbar buttons can be left-aligned in the layout (for functional buttons) and right-aligned for options and Save buttons.

Perhaps worth noting is that a web view can only render content that is already available in the SYSPRO application. For this reason, it is recommended that you first create the application and all the controls you may need before you design a web view.

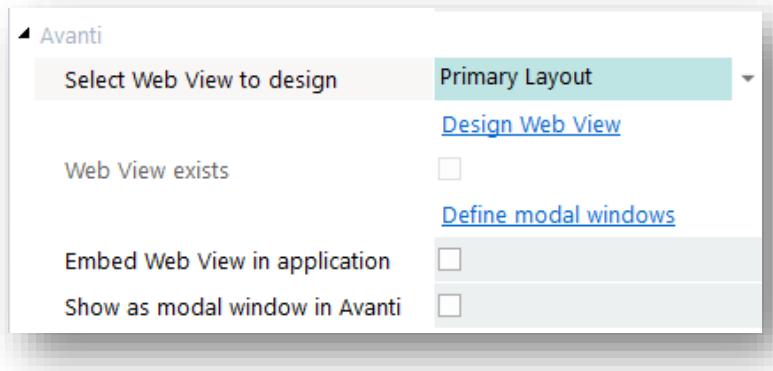
Having said this, you can always design and change the web view whenever you want to add, change, or remove layout components.

Note: Certain controls are currently not supported in a web view:

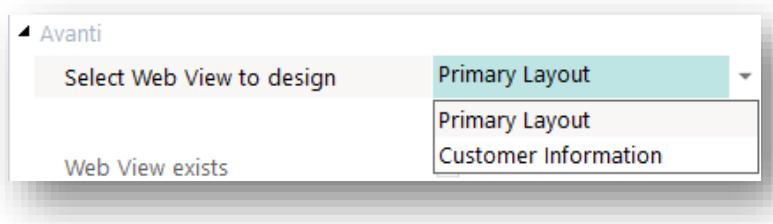
- Flowgraph
- Tile
- XAML Controls

DESIGNING A WEB VIEW

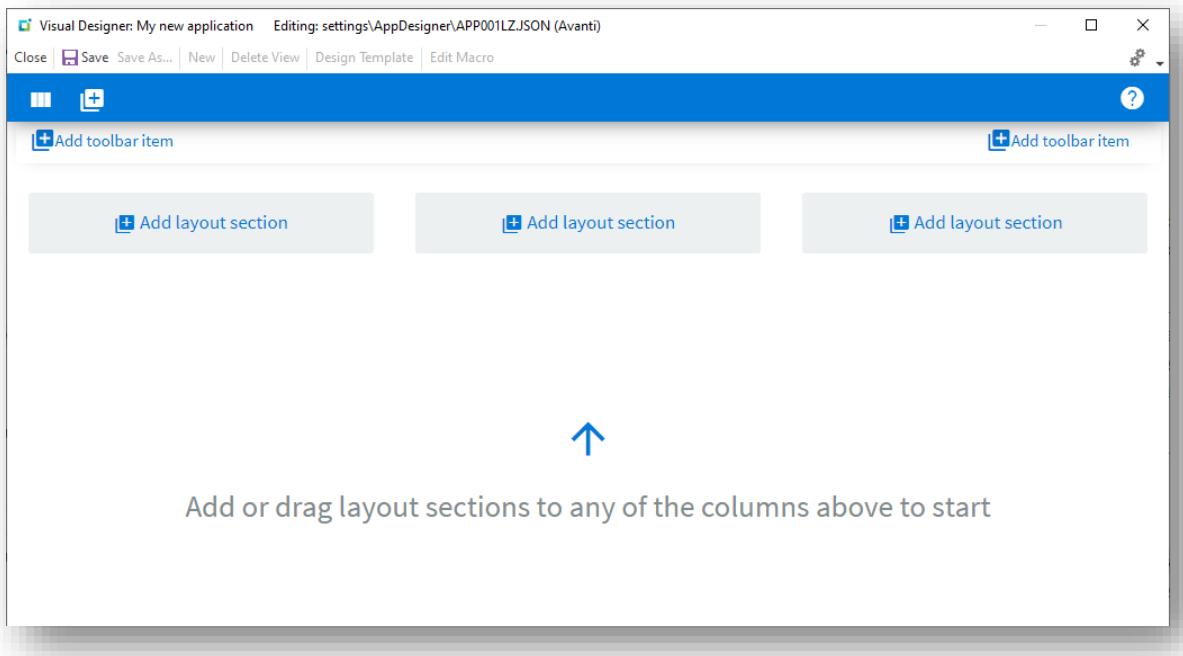
In the Application Details pane, you will see these options that are specific to Avanti:



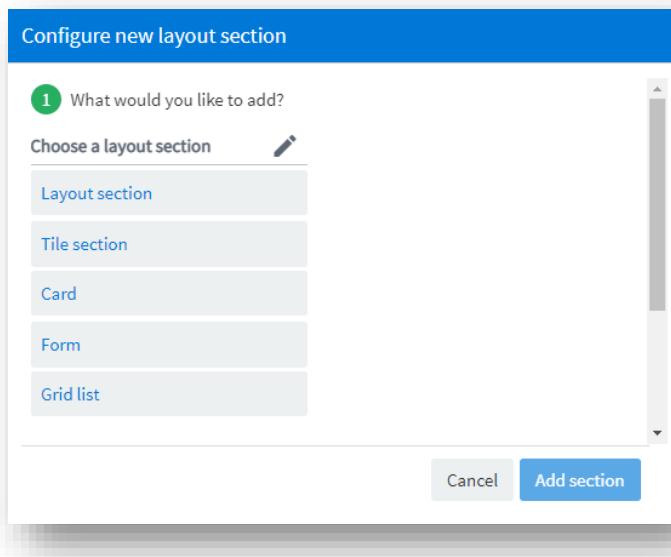
All web views have a 'primary layout' and optionally one or more modal windows. Any pane that is defined as a 'dialog box' will automatically be shown in the dropdown list, as will any additional modal window definitions.



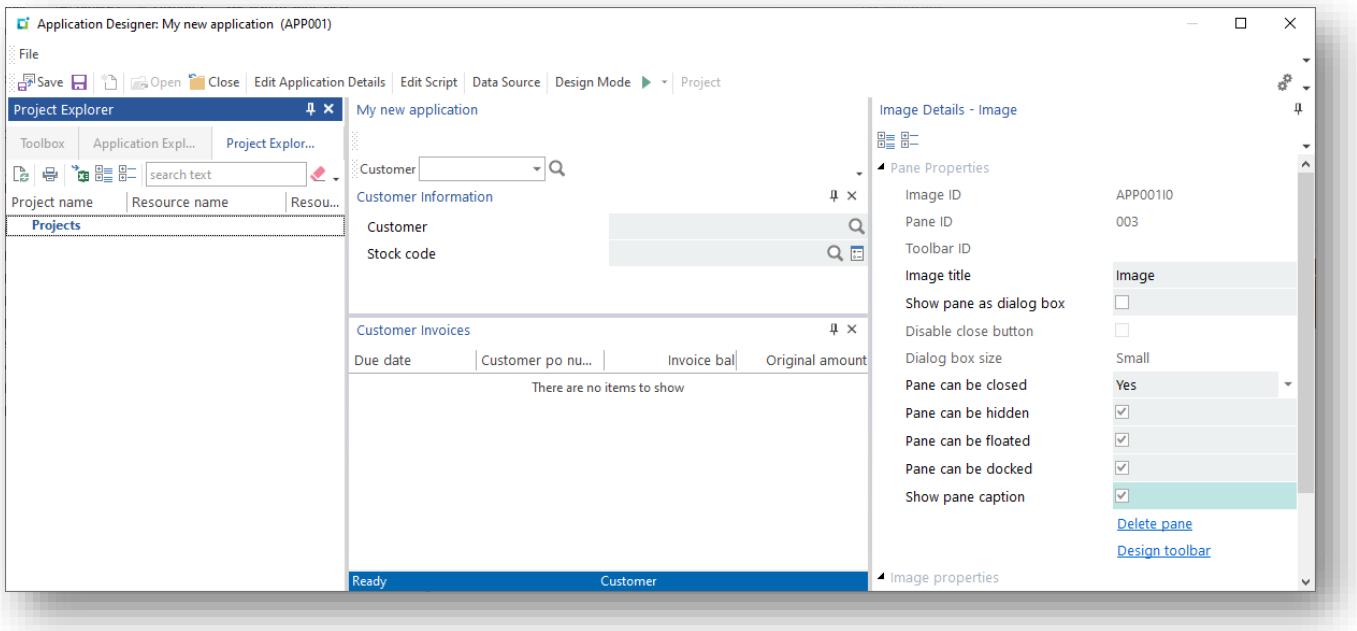
Select the web view you want to design and click on **Design Web View**. The Visual Designer is launched:



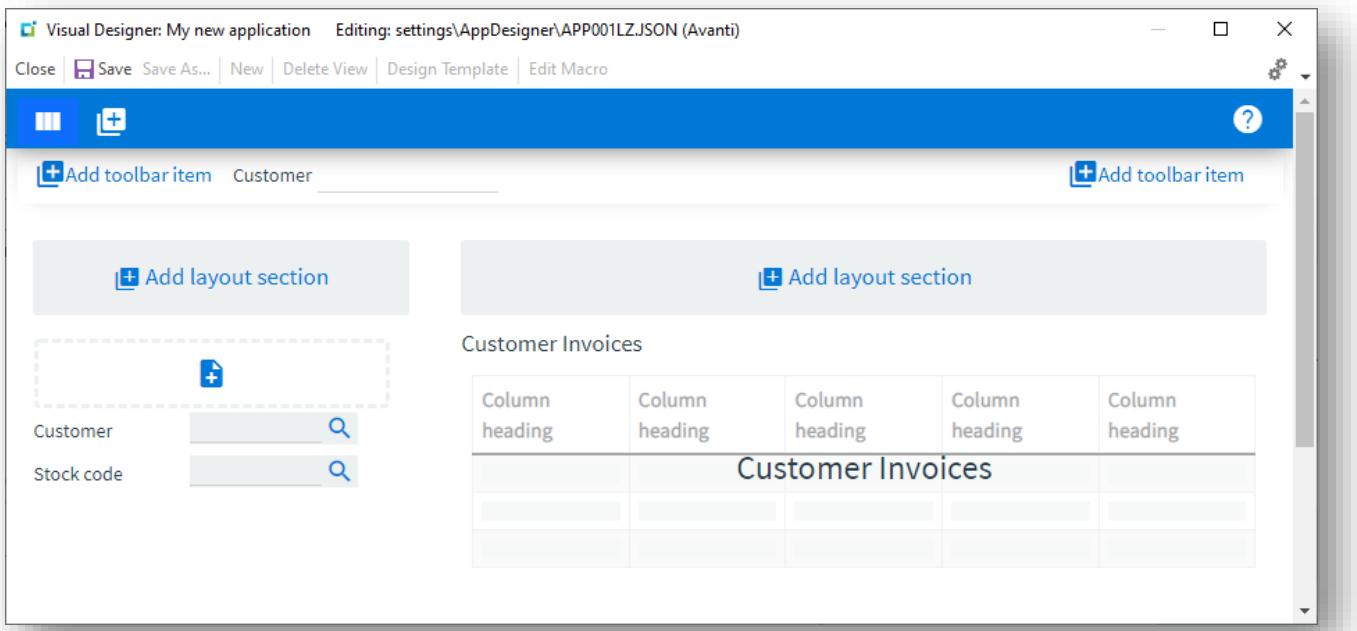
You can now design how you want your web view to look by adding toolbar buttons and layout sections:



As an example, here is a simple application that has a toolbar, a form and grid:



Here's how this application could be designed in the Visual Designer:



And here's what it looks like when running the application in SYSPRO:

The screenshot shows a window titled "My new application". At the top left is a dropdown menu labeled "Customer 0000001" with a search icon. To the right are settings and close buttons. Below the header is a section titled "Customer Information" with fields for "Customer" and "Stock code", each with a search icon. The main area is titled "Customer Invoices" and contains a table with the following data:

Due date	Customer po number	Invoice bal	Original amount
2015-03-11	BB801	447,998	448,000
2015-04-04	BB 805	914,000	914,000
2015-05-01	BB 809	584,800	584,800
2015-06-07	N900101-2004	2,110	2,110
2015-06-07	N900101-2004	784	784
2015-06-07	N900103-2005	1,190	2,390
2015-06-07	N900805-2005	16,493	16,493

At the bottom of the grid, it says "Customer: 0000001".

And then in Avanti:

The screenshot shows a browser window titled "My new application - SYSPRO Avanti". The address bar shows "localhost/_SYSPRO_Fusion/Staging/TestHarnessReleaseBC.html". The page header includes the SYSPRO Avanti logo, a navigation bar with "Home", and a user profile for "Phil Duff". Below the header is a search bar with "Customer 0000001" and a search icon. To the right is a "Customer Invoices" section with a table:

Due date	Customer po number	Invoice bal	Original amount
2015-03-11	BB801	447,998	448,000
2015-04-04	BB 805	914,000	914,000
2015-05-01	BB 809	584,800	584,800
2015-06-07	N900101-2004	2,110	2,110
2015-06-07	N900101-2004	784	784

At the bottom of the grid, it says "Showing 22 item(s)".

SHOWING THE APPLICATION AS A MODAL WINDOW

By default, when an application loads, the window will be maximized in the web browser.

This has the advantage of being able to use the full width and height of the web browser. However, if the application's initial window size is defined as 'small', then it might be appropriate to enable the option **Show as modal window in Avanti**.

SHOWING MODAL WINDOWS

When a Dialog Box is shown using the **ShowDialogBox** function, the IDE will automatically issue an Avanti **ShowModalWindow** statement.

When the Dialog Box is closed the Avanti, the **CloseModalWindow** statement is executed.

Thus, Dialog boxes in SYSPRO work identically in Avanti.

Note: The name of the web view for a dialog box will be generated as `UX_APP_NameOfControl.JSON` and saved in the `Settings\Appdesigner` folder.

FORM PROPERTIES

In the Form details pane, there is an option **Disable auto save**.

This option should be checked if the form's values should not be automatically saved when running in Avanti.

Examples of ensuring this is checked are month-end programs (e.g. ARSP01, etc.) as these types of programs should not have form values automatically saved upon exiting the application.

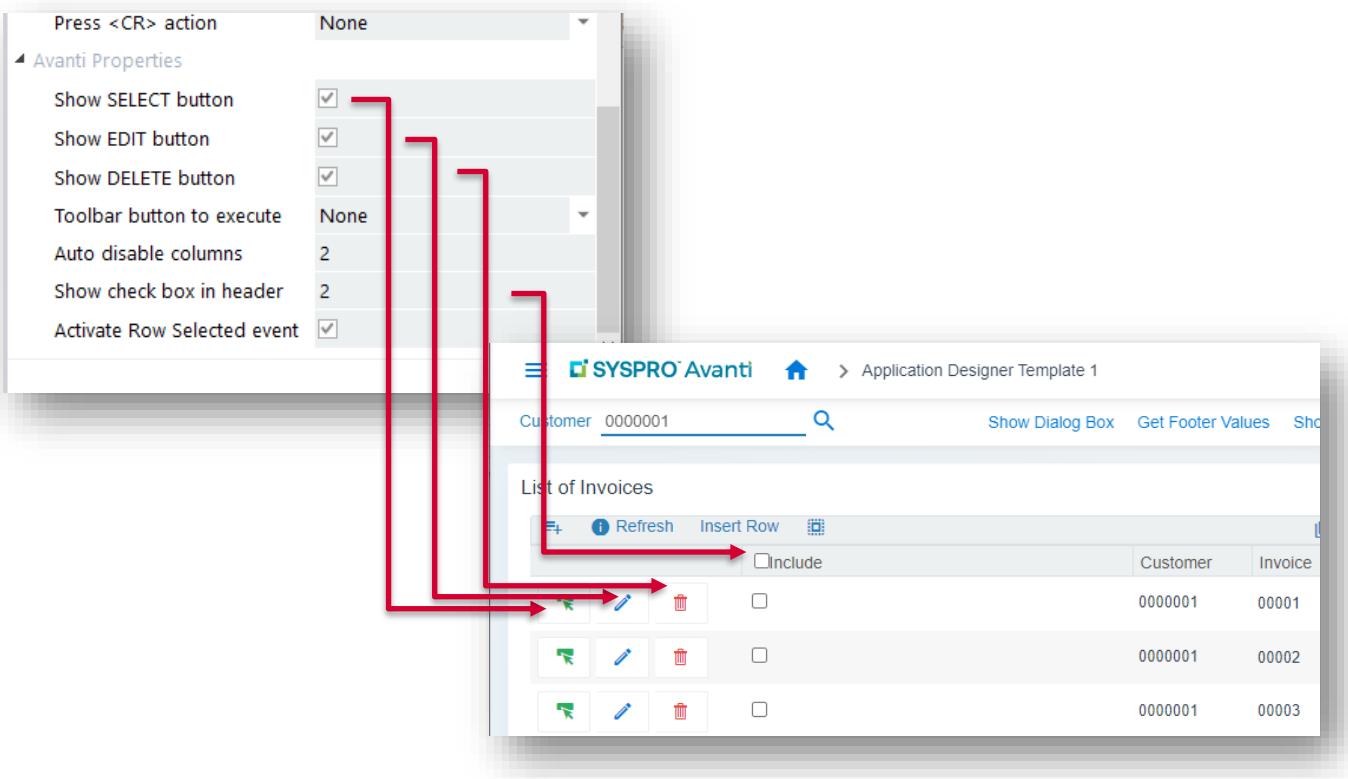
DESIGNING LIST VIEWS FOR AVANTI

In the List view details pane, you will find the **Avanti Properties**.

These options further control the behavior specifically for an Avanti grid:

Property	Explanation
Show SELECT button	Shows a SELECT button.
Show EDIT button	Shows an EDIT button.
Show DELETE button	Shows a DELETE button.
Toolbar button to execute	The toolbar button to execute when the DELETE button is clicked.
Auto disable columns	Columns to be automatically disabled when cell value changes.
Show check box in header	Show check box in column header(s).
Dynamic combo box columns	Columns that contain dynamic combo boxes.
Hyperlink columns	Columns that contain hyperlinks.

For example:



CREATING ADDITIONAL MODAL WINDOWS FOR AVANTI

A modal window is automatically shown in Avanti for any pane within an application that is defined as a dialog box when the callout **ShowDialogBox** is executed. However, there are times when you need to show a modal window in Avanti that's not specifically tied to a dialog box.

Typically, this is when the layout in a web view is of necessity quite different to that in the SYSPRO Windows version.

To create additional modal windows and have them shown at appropriate times when running an application requires the following:

1. Define the modal window properties in the Application Designer.
2. Design the web view for the modal window.
3. Show the modal window using the callout **ShowModalWindow**.
4. Close the modal window using the callout **CloseModalWindow**.

DEFINING MODAL WINDOWS

In the **Application Details** properties, click on the link **Define modal windows**:

The screenshot shows the 'Application Details' properties window. On the left, there's a tree view with 'Properties' expanded, showing 'Author', 'Release version', 'Last modified' (set to '02/07/2022'), and 'Show notes in About window'. To the right, under 'Primary Layout', there's a section titled 'Select Web View to design' with 'Design Web View' checked. Below this, there are two options: 'Embed Web View in application' and 'Show as modal window in Avanti'. A red box highlights the link 'Define modal windows' located in a separate section below these options.

Define any additional modal windows.
Modal windows are shown with the ShowModalWindow callout which requires the modal window name (the name of the application plus the 2-character modal ID). Each modal window has a window title and a WebView name (this is the name of the JSON webview file). The modal type can be:
No CLOSE button: A CLOSE button to close the window must be added to the window's toolbar.
Auto CLOSE: Show the window as modeless with a CLOSE button.
Show CLOSE button: A CloseModalWindow callout is required to close the window.

Enter up to 5 modal windows:

The screenshot shows the 'Avanti Modal Windows' dialog box. It has a search bar at the top and a table below with columns: 'Modal window ID', 'Window title', 'WebView name', 'Modal type', and 'Initial window size'. There are two entries: F0 (Window title: Proof of delivery, WebView name: Delivery, Modal type: Auto CLOSE, Initial window size: Medium) and F1 (Window title: Create sales orders, WebView name: SaleOrders, Modal type: Auto CLOSE, Initial window size: Medium).

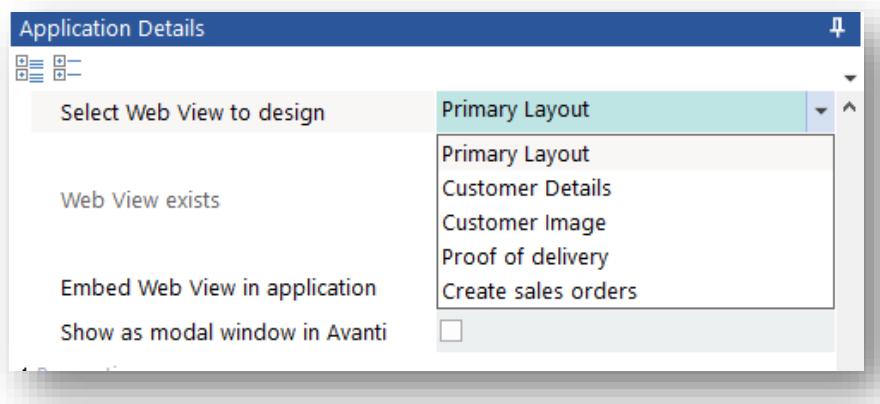
Modal window ID	Window title	WebView name	Modal type	Initial window size
F0	Proof of delivery	Delivery	Auto CLOSE	Medium
F1	Create sales orders	SaleOrders	Auto CLOSE	Medium

Column	Explanation
Modal window ID	<p>The Modal window ID must be a 2-character ID. When you show a modal window using the ShowModalWindow callout, you must specify the 'modal name' which comprises the application name and modal window ID.</p> <p>Therefore, once you have used the ShowModalWindow in your code, you must change your code if you subsequently change this modal window ID.</p>
Window title	The Window title must be non-blank and is shown as the title of the modal window.
Web View name	<p>The Web View name defines the name of the Avanti JSON file associated with this modal window.</p> <p>It must be non-blank and cannot contain any special characters (/ ?: & \ * " < > # % or space).</p>
Modal type	<p>The Modal type describes how the modal window is to be 'closed': Modal windows can be shown with or without a CLOSE button (i.e. the X at the top right-hand corner of the window).</p> <p>The modal type may be one of the following:</p> <ul style="list-style-type: none"> ▪ NoCloseButton No CLOSE button is shown. The only way to close this type of modal window is to issue the CloseModalWindow callout. This is used when some logic needs to be performed before the window is closed. Note that the window is shown 'modal': <div style="text-align: center;"> <p>The screenshot shows a modal window with a light gray header bar containing the title 'Proof of delivery'. On the right side of the header are three small icons: a minus sign, a square, and an 'X'. Below the header is a horizontal line. Underneath the line, there is a 'CLOSE' button followed by a blue gear icon. The main body of the window contains three input fields labeled 'Customer', 'Customer name', and 'Branch', each with a magnifying glass icon to its right.</p> </div> <ul style="list-style-type: none"> ▪ AutoClose A CLOSE button is shown. Clicking on the X will automatically close the window. The window is shown 'modeless', meaning that the user can click behind the window: <div style="text-align: center;"> <p>The screenshot shows a modeless window with a standard window title bar containing the title 'Proof of delivery' and standard window controls (minimize, maximize, close). Below the title bar is a horizontal line. Underneath the line, there is a 'CLOSE' button followed by a blue gear icon. The main body of the window contains three input fields labeled 'Customer', 'Customer name', and 'Branch', each with a magnifying glass icon to its right.</p> </div>

	<ul style="list-style-type: none"> ▪ ShowCloseButton A CLOSE button is shown. Clicking on the X will trigger the event ModalWindowClose and then the CloseModalWindow callout can be executed to close the window. This is used when some logic needs to be performed before the window is closed. Note that the window is shown 'modal'.
Initial window size	The Initial window size may be Small, Medium or Large and specifies the default window size when it is initially shown.

DESIGNING A WEB VIEW FOR THE MODAL WINDOW

To design a web view for a modal window, select the modal window from the drop-down list in **Select Web View to design** and then click **Design Web View**.



SHOWING OR CLOSING A MODAL WINDOW

The additional modal windows are not automatically shown. You must decide in your code the appropriate time to show a modal window.

Use the callout **ShowModalWindow** with the modal name comprising of the 6-character application name and the 2-character modal ID.

Unless the modal window is defined as **AutoClose**, you will have to issue the **CloseModalWindow** callout in your code to close the modal window using the same modal name.

IMPROVING THE PERFORMANCE OF AN AVANTI APPLICATION

If a script in an application issues multiple callouts (such as enabling or disabling multiple toolbar fields or form fields) then this can impair the performance of an application running in Avanti. To overcome this problem, a script can issue a special callout

BeginWebViewUpdate. This callout helps improve the performance when running in Avanti.

SYSPRO programs refresh forms, toolbars, listviews and charts one by one. Each time this happens a call is sent to the web browser to refresh various fields in the web view. While only actions on fields in forms and toolbars that are actually used in the web view are sent to the web browser, nevertheless there is latency in each call.

To improve the performance of this you can inject into your script a callout to prevent any web view updates happening between the **BeginWebViewUpdate** and the **EndWebViewUpdate** callouts:

Optimize Avanti performance	(BeginWebViewUpdate, " ", " ")	Prevents any callouts being sent to Avanti until the 'EndWebViewUpdate' callout. This mechanism improves the performance of refreshing a Web View. The callout will be ignored if not in Avanti.
Commits Avanti callouts	(EndWebViewUpdate, " ", " ")	When the EndWebViewUpdate callout is executed any forms, toolbars or charts that were requiring to be refreshed will now be refreshed. The callout will be ignored if not in Avanti.

When the **BeginWebViewUpdate** callout is executed any form, toolbar or chart that is 'refreshed' is ignored from being passed onto the web view. Instead, when the **EndWebViewUpdate** callout is executed any forms, toolbars or charts that were requiring to be refreshed will now be refreshed. Additionally, updating of rows in a grid will only occur once.

Note that these callouts will only be executed when running in Avanti.

USING DATA SOURCES

The purpose of data sources is to allow you to create applications that obtain data from a variety of sources, and optionally bind data to Forms, List views or Charts. All this without having to do any code.

Data sources generally work as follows:

1. You create a data source that points to a SYSPRO table, SYSPRO business object, Custom data source or Web Service.
2. You add fields to a form (or columns to a listview or data points for a chart) and in so doing define which data source is to be applied.
3. You decide at which point in the application that one or more data sources should be executed.
It's important to note that most data sources (if not all) would require some sort of key (a Customer number, for example) to be applied.

You can also create data sources that are transactional rather than Query. Transactional data sources allow you to define data sources that update the SYSPRO or any external database using either a SYSPRO Business Object or a SQL statement. Please read the section

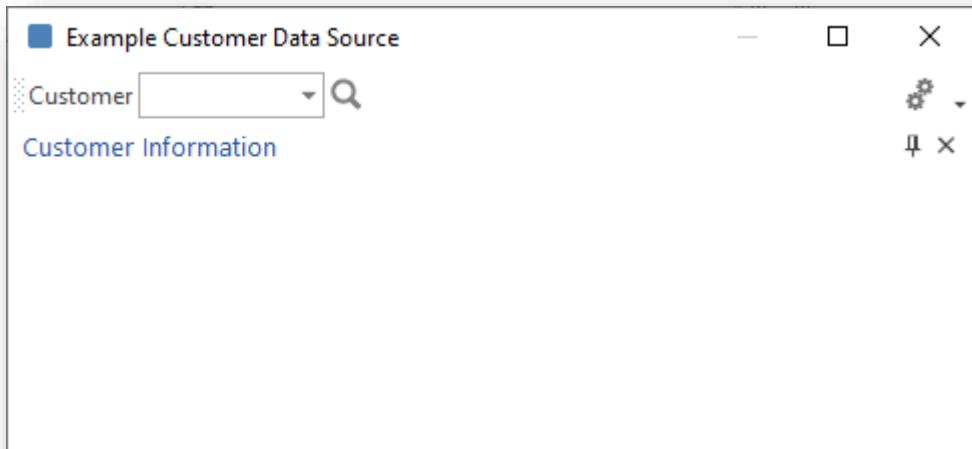
Transactional Data Sources for more information.

CREATING A QUERY DATA SOURCE

Let's see how this works in an example:

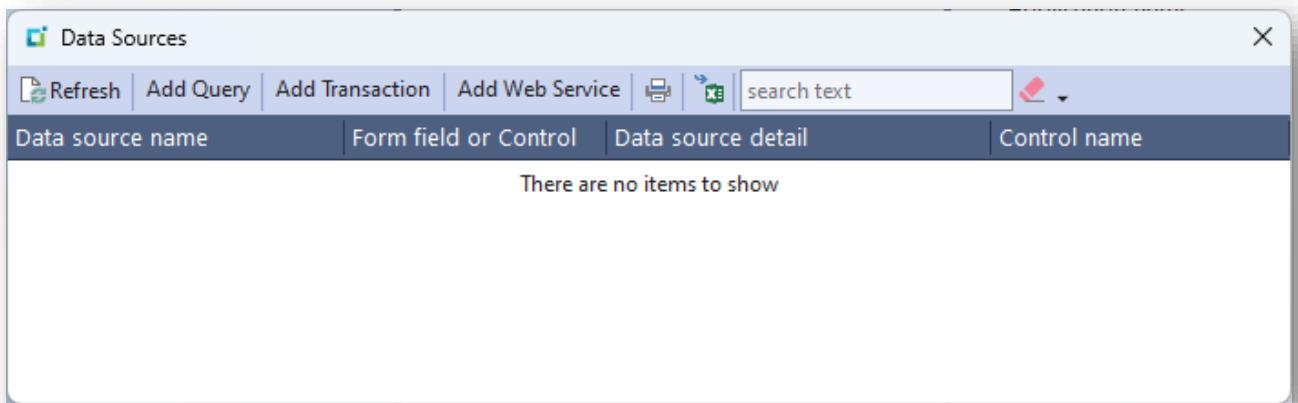
In an application, you enter a customer number in the main toolbar and then we want various form fields to be automatically updated with information derived from the Customer master table.

To simplify this explanation, let's assume we have this outline application that contains a Customer toolbar field and a blank form:

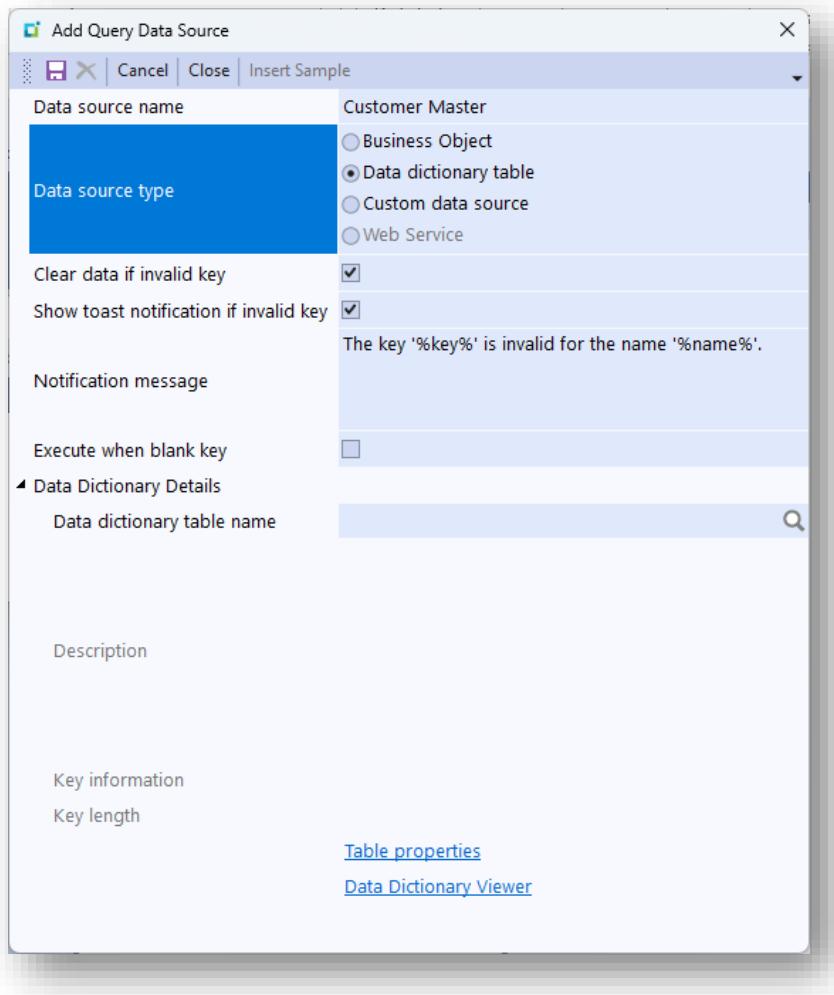


STEP 1: CREATE A DATA SOURCE

From the toolbar, click Data Sources:



And then click Add Query. The data source window is displayed:



Each data source must have a unique name by which it is identified.

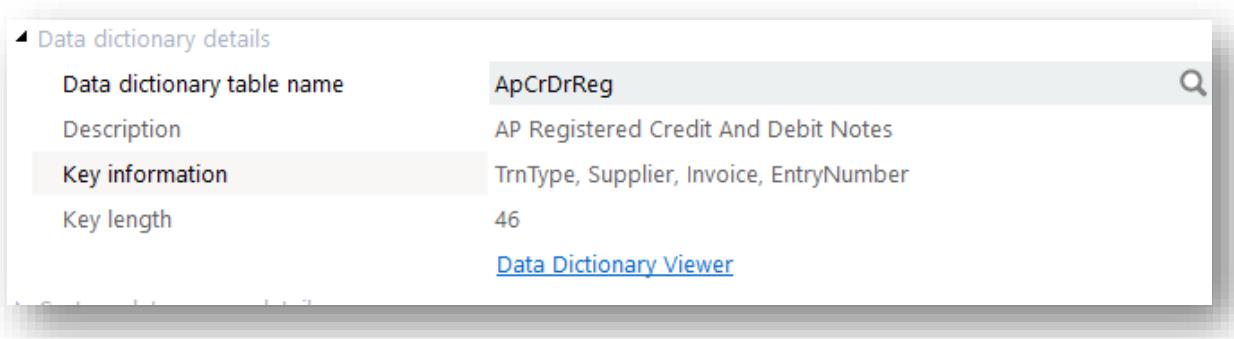
- This can be up to 30 characters.
- Once created, you cannot change the name of a data source.
- In the example above, we have entered **Customer Master**.

There are various types of data sources (Data dictionary table name, Business object, Custom data source).

- For this exercise we have selected **Data dictionary table**.

In the **Data dictionary table name** field, you can start typing the name and predictive search will help you identify the correct table. You can also press the browse button to see all available tables.

- In the example above, we have entered **ArCustomer**.
- Some tables may require a key constructed of multiple elements. This information is shown in the Key information property as column names separated by a comma:



You can also click on **Data dictionary viewer** to view all tables and their properties.

Optional configuration for the scenario of an invalid key being entered:

- You can define if your form field values should automatically be cleared.
- You can indicate if a toast notification should be displayed and define the contents of the Notification message.
 - The message can optionally contain the placeholders:
 %key% – the key supplied to the data source
 %name% – the table or business object used in the data source
 %msg% – the original error message returned from a business object
 - This message is shown only if the option **Show toast notification if invalid key** is switched on.

Execute when blank key. By default, this option is unchecked. This indicates that the data source will not be executed if the KEY value to be applied is blank. You will need to check this option if a data source can be executed without a KEY having to be supplied.

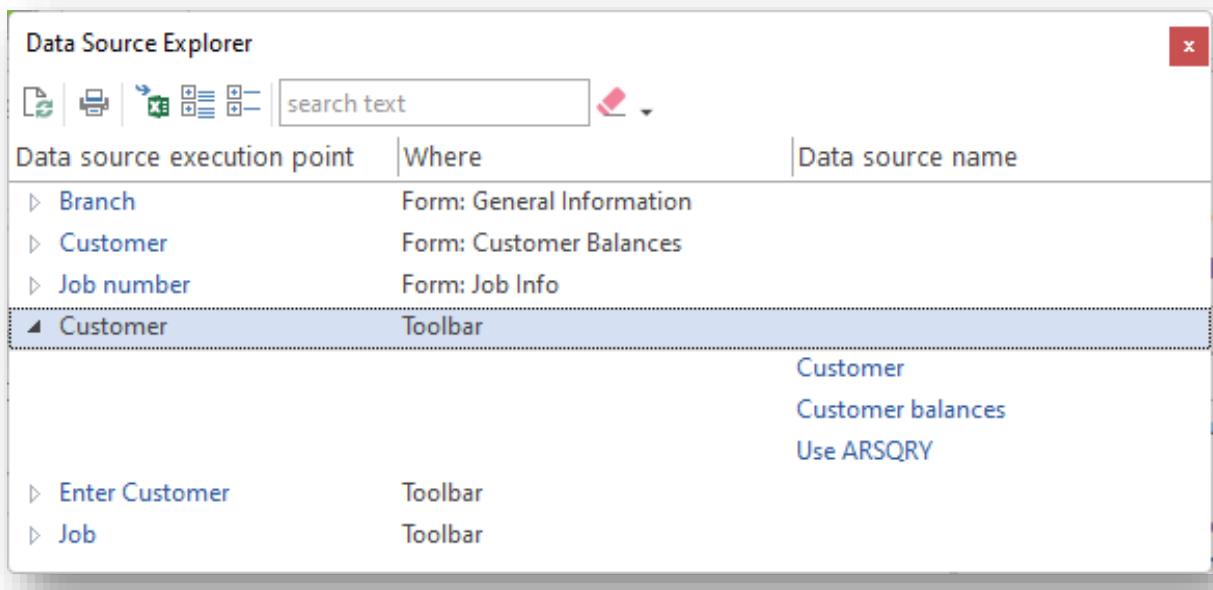
You can change these options later if required.

Note: When a data source is executed after a toolbar button, form field or listview cell value has changed (and if the KEY supplied to the data source is invalid) then focus will be automatically set to that toolbar, field or cell. This helps the user re-enter a valid key.

However, this doesn't apply to a data source that is executed from a script callout.

Click the **SAVE** button, followed by the **CLOSE** button.

In the **Data Source Explorer** you can see where a data source will be executed in the application:



STEP 2: ADD FIELDS TO A FORM USING A DATA SOURCE

We can now add fields to the form using the data source we have just created.

Click on the Customer Information pane and you will see the **Form Details** and **Form Field Properties** panes are displayed.

The screenshot shows the Application Designer interface with the following components:

- Example Customer Data Source**: A pane on the left showing a dropdown menu set to "Customer".
- Form Details - Customer Information**: A pane on the right with a red border, containing:
 - Bind Properties**: A section with a dropdown menu "Select fields from data source" which has "Customer Master" selected.
 - Form Field Properties**: A section with fields:
 - Caption
 - Override description
 - Type
 - Length: 0
 - Decimals
 - Field binding
 - Data sources**: A section with a link "Data source to execute".
- Data source list**: A link at the bottom of the Form Details pane.

From the **Select fields from data source** field, select **Customer Master** from the dropdown list. All the fields derived from the **ArCustomer** master table will be shown in a listview:

Add Selected Items	Description	Data type	Column name	Length	Deci...
<input type="checkbox"/>	Create notes allowed	Alpha	CRNOTESALLOW...	1	
<input type="checkbox"/>	Credit status code	Alpha	CreditStatus	1	
<input checked="" type="checkbox"/>	Customer	Alpha	Customer	15	
<input type="checkbox"/>	Customer P/o number mandatory	Alpha	PoNumberM...	1	
<input checked="" type="checkbox"/>	Customer class	Alpha	CustomerClass	10	
<input type="checkbox"/>	Customer currency code	Alpha	Currency	3	
<input type="checkbox"/>	Customer e-mail address	Alpha	Email	255	
<input checked="" type="checkbox"/>	Customer name	Alpha	Name	50	
<input checked="" type="checkbox"/>	Customer on hold	Alpha	CustomerOn...	1	
<input type="checkbox"/>	Customer type	Alpha	CustomerType	1	
<input type="checkbox"/>	Customer/stock code cross ref.	Alpha	MaintLastPrc...	1	
<input type="checkbox"/>	Date customer added	Date	DateCustAdd...	8	
<input type="checkbox"/>	Date of last payment	Date	DateLastPay	8	
<input type="checkbox"/>	Date of last sale	Date	DateLastSale	8	

You can now select which fields you want to add to the form, in one of two ways:

- Click on a hyperlinked description (this will immediately add the item to the form).
- Check which items are to be added in the **Select** column and click **Add Selected Items**.

In the example above, the *Customer*, *Customer class*, *Customer name* and *Customer on hold* fields have been selected.

Click on **Add Selected Items** and the form will look as follows:

The screenshot shows the Application Designer interface. On the left, the 'Example Customer Data Source' pane displays a list of fields: Customer, Customer class, Customer name, and Customer on hold. The 'Customer' field is currently selected. On the right, the 'Form Details - Customer Information' pane is open, showing the properties for the selected field. The 'Caption' is set to 'Customer', 'Type' is 'Alpha', and 'Length' is '15'. The 'Field binding' is set to '[Customer Master]Customer'. A link labeled 'Data source to execute' is also present in this pane.

As you click on a form field, so its properties are displayed. You will notice that there is a **Field binding** property that indicates if the field is bound to a data source. This field can be edited to change the data source name and binding field name, but currently there's no validation if this binding property is changed.

Considerations:

- The binding property can also be seen when designing a form in the **Tag** property, although there the property is read-only.
- You can change the form field's caption or Override description from this window.

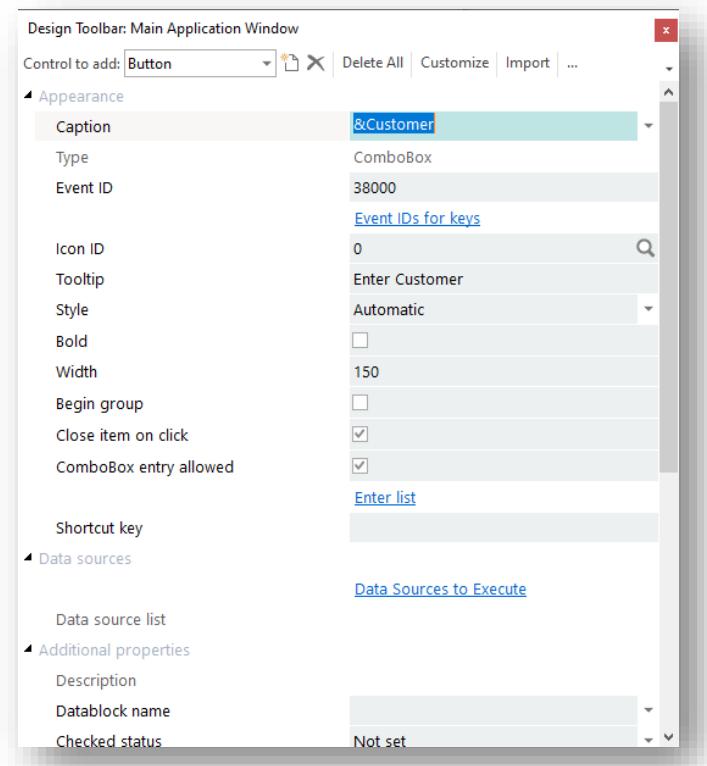
STEP 3: EXECUTE THE DATA SOURCE

At this point, the application has a form with fields bound to a data source. But there's no indication yet as to how the data source will be executed to update the form field values.

There are several ways of executing a data source:

- When a toolbar value changes
- When a form field value changes
- When a listview row is 'selected'
- When a listview cell value changes
- When clicking on a chart data point
- Executing a data source from script

To execute the data source when a toolbar value changes, select to Design the toolbar, and then select the **Customer** control:



Select the **Data Sources to Execute** hyperlink and a window is presented indicating all the data sources. Note that the selected data sources will be executed in the sequence presented in this listview, so if required you can drag a data source up or down the list to reposition it:

Select Data Sources			
Select	Data source name	Data source type	Detail
<input checked="" type="checkbox"/>	SQL Chart from Customer	Custom data source	OleDb Provider
<input checked="" type="checkbox"/>	Drill down for invoice list	Custom data source	OleDb Provider
<input checked="" type="checkbox"/>	Chart SQL	Custom data source	OleDb Provider
<input type="checkbox"/>	Monthly Sales	Custom data source	OleDb Provider
<input type="checkbox"/>	Warehouse Points	Custom data source	OleDb Provider

When done, click **Apply**, and the data source list will now reflect this:

The screenshot shows a navigation pane on the left with 'Data sources' expanded. Below it, a blue bar contains the 'Data Sources to Execute' link. Underneath, the 'Data Sources to execute' section lists three items: 'SQL Chart from Customer', 'Drill down for invoice list', and 'Chart SQL'.

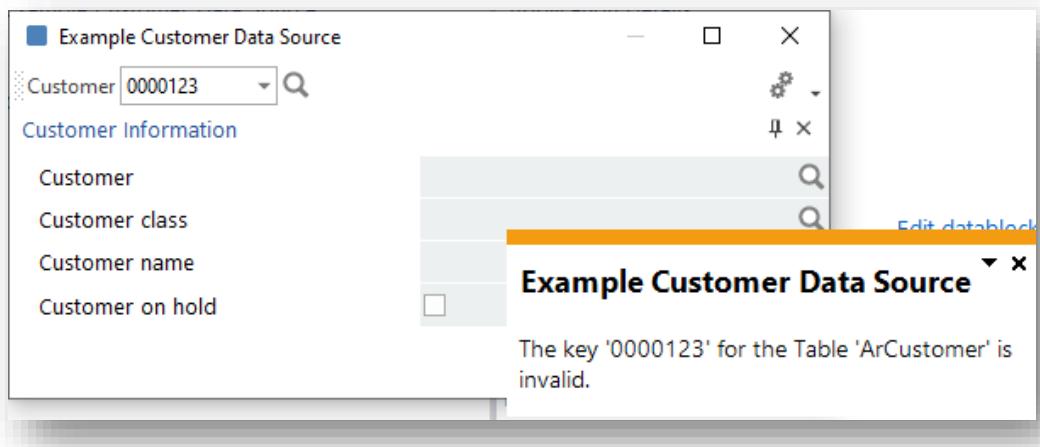
STEP 4: RUN THE APPLICATION

You can now run the application by pressing F5.

Enter a customer number and your form fields will be refreshed with values derived from the Customer Master table:

The screenshot shows a form titled 'Example Customer Data Source'. At the top, there is a dropdown menu labeled 'Customer' with the value '0000002' and a search icon. Below the dropdown are sections for 'Customer Information' and 'Customer Details'. The 'Customer' field has the value '0000002'. The 'Customer class' field has the value 'M'. The 'Customer name' field has the value 'Bikes & Blades - North'. The 'Customer on hold' field has a checked checkbox. There are also settings and refresh icons at the top right of the form.

Now try entering a customer that doesn't exist:

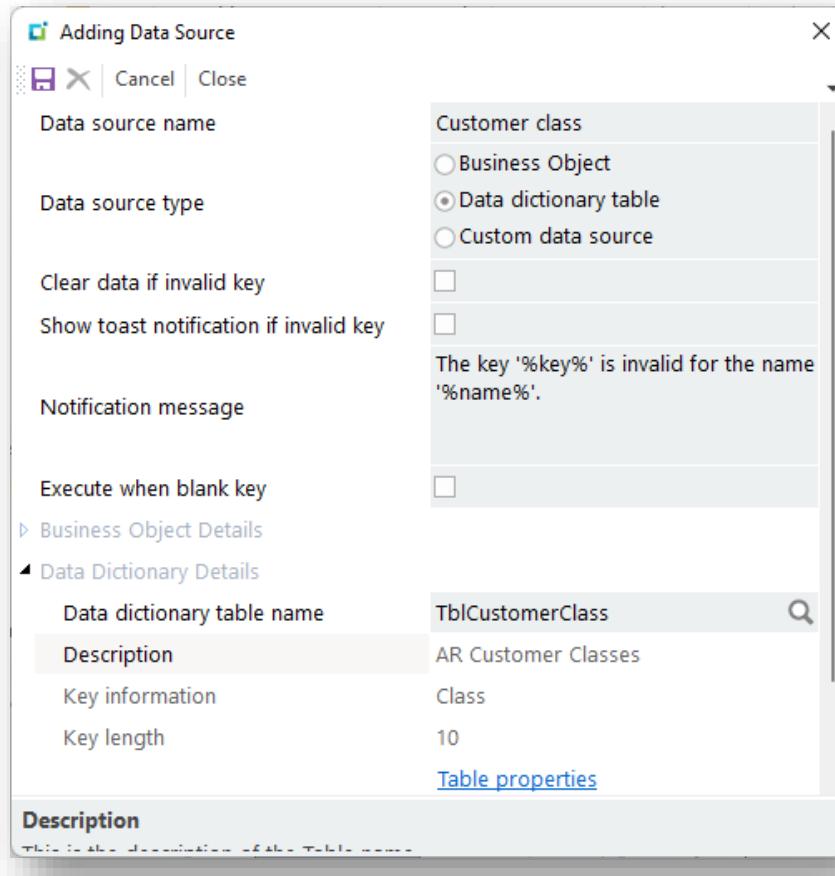


You will see a toast notification indicating the invalid key, and you will notice that the form field values have been cleared.

HOW TO SHOW THE DESCRIPTION OF A 'KEY' ITEM IN A FORM

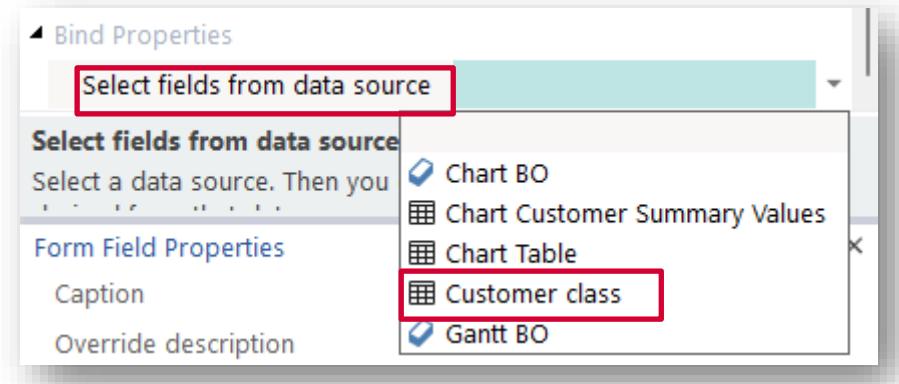
In our application we have a **Customer class** form field. It would be useful to show the Customer class description beneath this field. Here's how to do that:

First, create another data source using the table `TblCustomerClass`:



Now we can add the Customer class description field to the form.

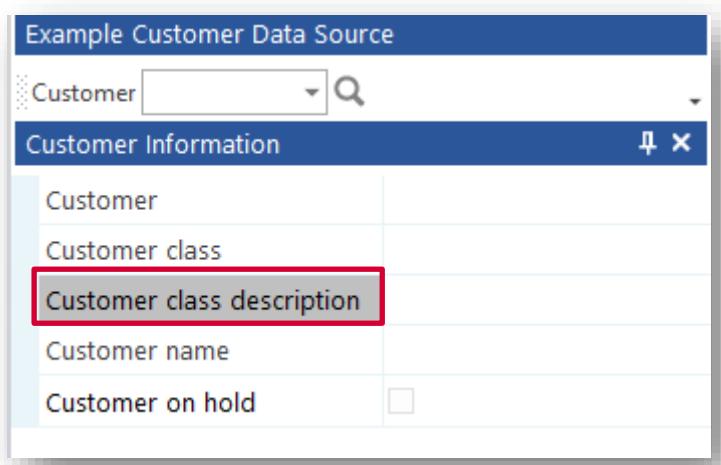
Select **Customer class** data source:



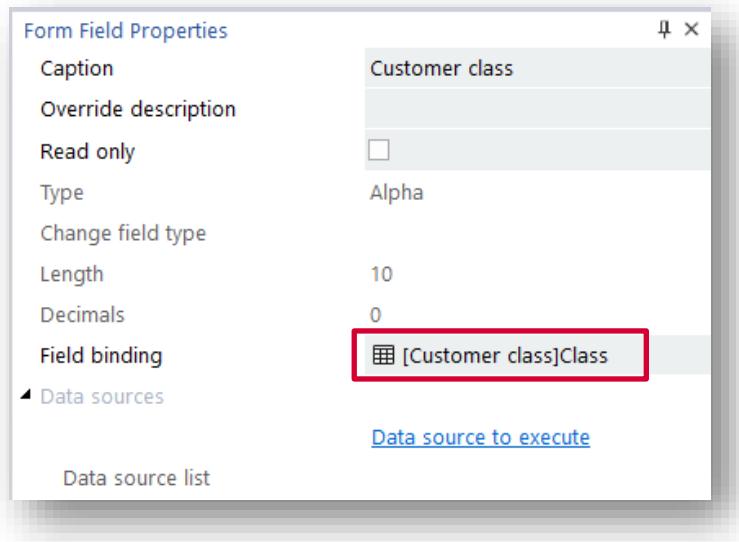
And then select the **Customer class description** field:

Fields from Data Source: Customer class (TblCustomerClass)				
Add Selected Items	search text	Data type	Column name	Leng...
<input type="checkbox"/>	Customer class	Alpha	Class	10
<input checked="" type="checkbox"/>	Customer class description	Alpha	Description	50

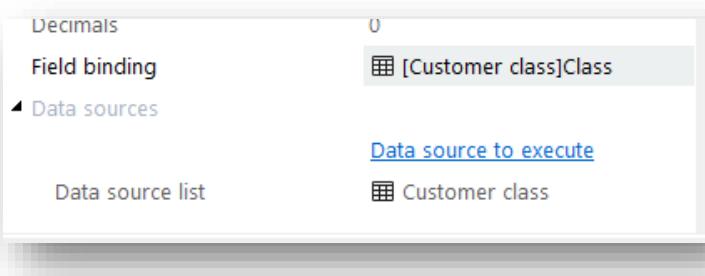
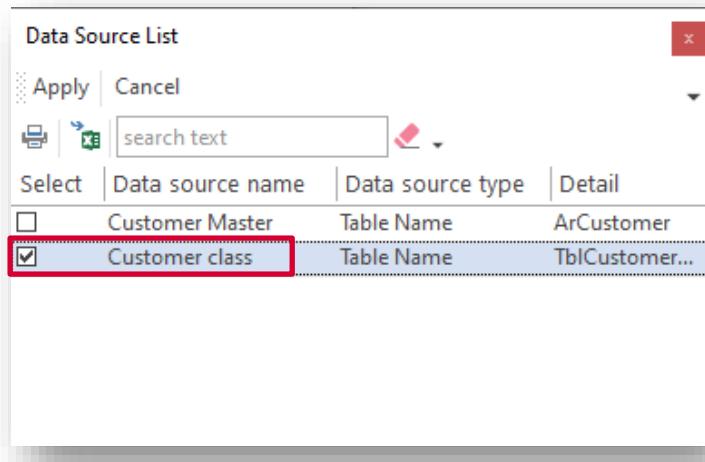
Drag the **Customer class description** to be beneath the **Customer class** field:



And finally, click on the **Customer class** form field and select the **Data source to execute** function:



Select the **Customer class** data source and click **Apply**:



Now run the application by pressing F5.

Enter a customer number and then type a new customer class into Customer class field – you will see that the class description is now refreshed:

The screenshot shows a 'Customer Information' form window titled 'Example Customer Data Source'. It contains the following fields:

Field	Value
Customer	0000001
Customer class	B
Customer class description	Bad Debts
Customer name	Bayside Bikes - a very long name
Customer on hold	

Note: When a form field value is changed (either by the application's script or because a data source was executed which in turn changed a form field value) then any data source associated with that field will be executed.

HOW TO APPLY A TOOLTIP (KEY DESCRIPTION) TO A FIELD USING A DATA SOURCE

You can apply a tooltip to a form field using a data source without having to use any code. This is very useful if you wish to see the description of a KEY field in the form – in the example below, the **Contact name** is shown next to the Customer form field:

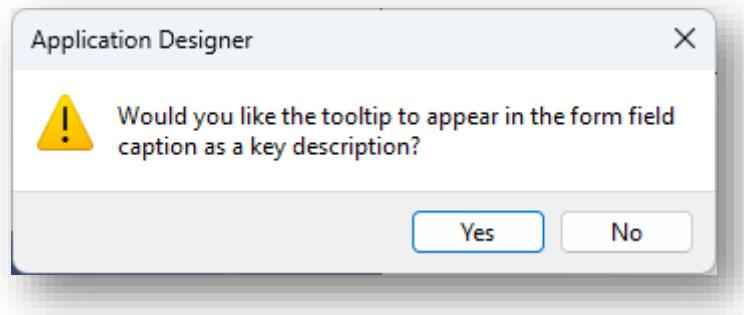
The screenshot shows a 'Customer Info' form window. The 'Customer' field has a tooltip displaying 'Susan Brown 0000002'.

In the FORM FIELD PROPERTIES window is a hyperlink [Select tooltip binding](#) – click this to select a column from the available columns derived from the data source defined in Field binding. In this example, the column **Contact** is selected:

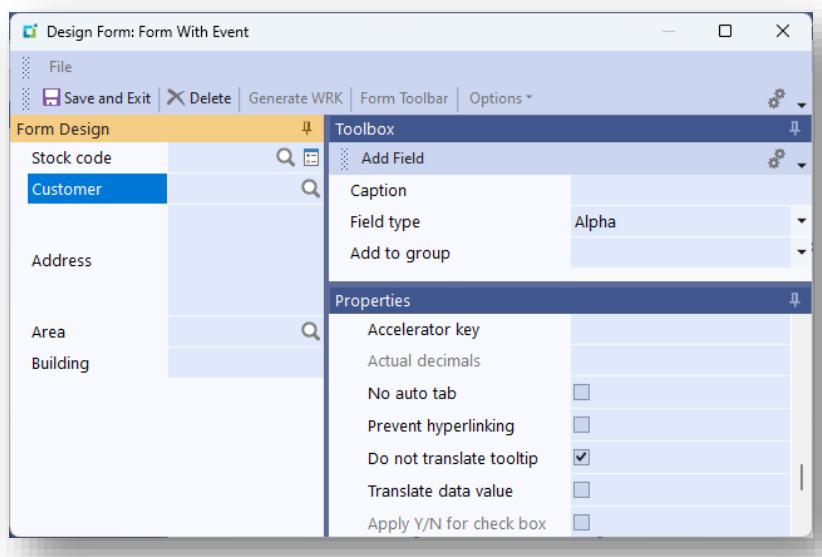
The screenshot shows the 'Form Field Properties' window for the 'Customer' field. Under the 'Data binding' section, the 'Tooltip binding' dropdown is set to 'Contact'.

The tooltip will be applied against the form field caption if the field property 'Do not translate tooltip' is checked on – this property states that the tooltip is not to be translated which is an indication that the tooltip is a 'description' of a key field.

If, when selecting a tooltip binding column, the 'Do not translate tooltip' property is unchecked then the Application Designer will prompt you if you wish to switch it on:



Here's a screen shot of this field property in Form Designer:



Note: Also look at the **Validate field on change** property.

HOW TO SHOW MULTIPLE COLUMN NAMES FOR A FORM FIELD

It is possible to define multiple column names for a field binding which is particularly useful if the form field is a multiline type. In the example below there is a multiline ADDRESS form field whose field binding consists of multiple column names (separated by a comma) all derived from the data source 'Ar Customer Table':



And it will appear like this in the form:

Customer	<u>0000001</u>
Address	3 Beach Road Bayside North Seattle WA

The multiple column name values are inserted into the form field with Line Feed characters between each value if the form field is of type multiline and with a space character if not multiline.

The multiple column names have to be entered manually against the FIELD BINDING property – if a column name is not named correctly it will be shown with the text ‘Invalid:’.

Additionally, a column name may be used as a tooltip simply by suffixing the column name with the words **(tooltip)**, as in this example:

[Ar Customer Table]Customer,Name(tooltip)

This will appear like this in a form field:

Customer	Bayside Bikes	0000001
Address	3 Beach Road Bayside Bikes Seattle WA	

EXECUTING A DATA SOURCE FOR A LIST VIEW

Row 'SELECTED' EVENT

When a row becomes 'selected' (i.e. becomes the focused row) you can execute a data source using the value of a specific column to be applied to that data source as the KEY. This might be useful if you wish to show some more information in a separate form using a data source.

The following is an example application that shows a list of INVOICES for a customer:

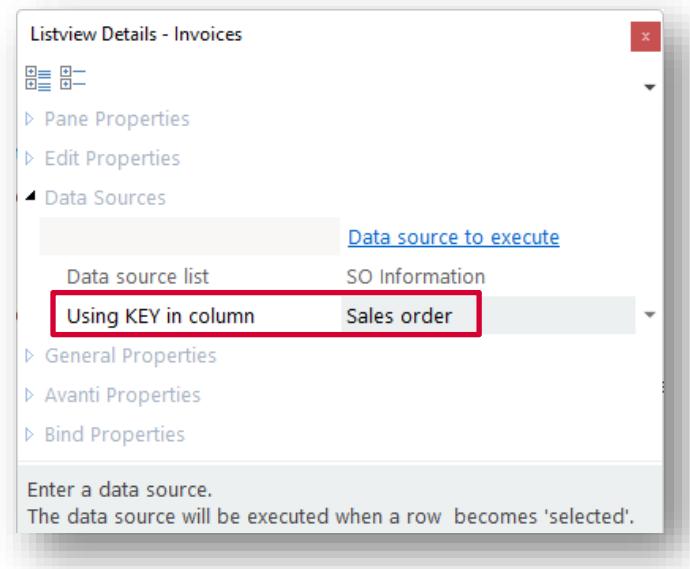
The screenshot shows a desktop application window titled "Listview Data Source App". On the left, there is a list view titled "Invoices" with columns: Sales order, Invoice, Invoice date, and Customer po number. The first row, which has a sales order of 000793 and an invoice of 100466, is highlighted. On the right, there is a separate form titled "Sales Order Information" with fields: Sales order number (000803), Customer (Customer 000001), Customer name (Bayside Bikes), Branch code (10), and Sales order date (05/03/2015). The "Customer" field in the "Sales Order Information" form is also highlighted, indicating it is the selected row from the list view.

When you click on a row, the information about the Sales Order is then shown in a separate form. In this example, the form fields in the form **Sales Order Information** are bound to the data source **SO Information**.

This screenshot is similar to the previous one, showing the "Listview Data Source App" window. The "Invoices" list view on the left has the same structure and data. The second row, with a sales order of 000803 and an invoice of 100476, is now highlighted. The "Sales Order Information" form on the right has updated values: Sales order number (000803), Customer (Customer 000001), Customer name (Bayside Bikes), Branch code (10), and Sales order date (05/03/2015). The "Customer" field in the "Sales Order Information" form is highlighted, indicating it is the selected row from the list view.

To achieve this using a data source:

1. Against the List view properties pane, select the appropriate **Data Source to Execute**. In this example, a data source of **SO Information** has been chosen (this points to the table **SorMaster**).
2. Select the name of the column in which the KEY to execute the data source is found, using the property **Using KEY in column**. In this example, we will use the SALES ORDER column:



And now, when a row is selected, the data source SO Information is executed using the KEY found in the column Sales Order, which in turn will populate the Sales Order Information form.

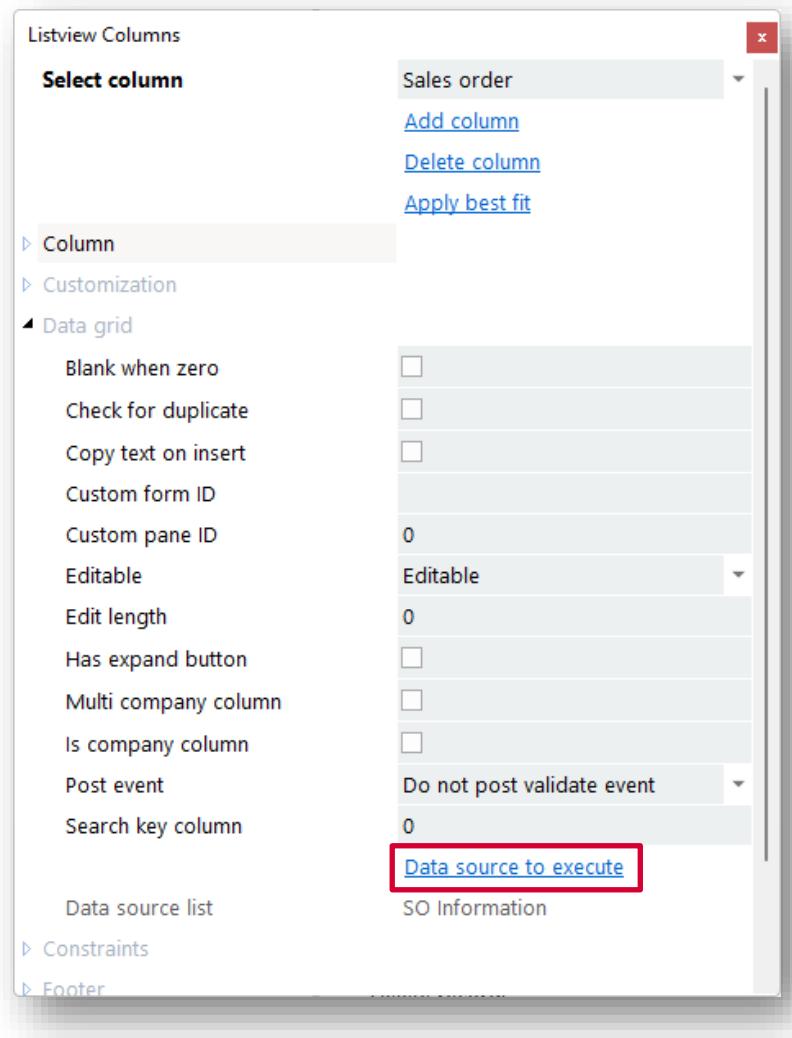
CELL VALUE CHANGED EVENT

When a cell in a listview is changed, you can execute a data source using the value of the cell to be applied to that data source as the KEY.

Typically, you would use this to validate the cell value. If the cell value is not a valid KEY then a toast notification is displayed, and focus is set back on the cell. If the cell value is a valid KEY, then the data source field bindings are executed.

To set this up, proceed as follows:

1. In the Listview Columns window, select the **Data Source to Execute** hyperlink and the required data source:



Note: The column must be defined as **Editable** for this to work.

So, using our example, enter a Sales Order in the cell and the Sales Order Information form will be populated:

The screenshot shows a Listview Data Source App window divided into two main sections. The left section, titled "Invoices", contains a table with four columns: Sales order, Invoice, Invoice date, and Customer po number. The data includes rows for sales orders 000813, 000801, 000813, and 000824. The right section, titled "Sales Order Information", displays five fields: Sales order number (000901), Customer (0000001), Customer name (Bayside Bikes), Branch code (10), and Sales order date (22/10/2020). A search bar at the top of the left pane shows the value "Customer 00000012".

This screenshot shows the same Listview Data Source App window after a sales order has been selected. The sales order number 000901 is now highlighted in the "Invoices" table. The "Sales Order Information" pane has updated to show the details for this specific sales order: Sales order number 000901, Customer 0000001, Customer name Bayside Bikes, Branch code 10, and Sales order date 22/10/2020.

EXECUTING A DATA SOURCE VIA CODE

You can also execute a data source via script.

This may be very useful where the application needs to refresh form field data at some point in the logic, rather than when a toolbar or field value changes. Additionally, the data source in question may require a compound 'key' to be passed to it – for example, a Stock code AND a Warehouse value.

In these cases, you can use two data source callouts, as indicated in the image below:

- **DataSourceExecute** callout:

This callout executes the named data source with a KEY and returns the data from that data source.

- **DataSourceRefresh** callout:

This executes the name data source with a KEY and automatically refreshes any form fields bound to that data source.

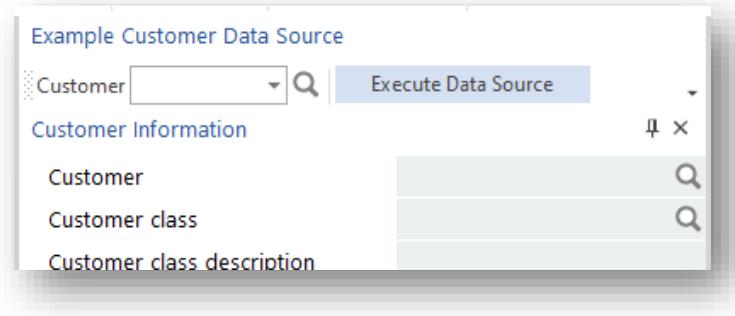
In both cases, if the KEY being passed is invalid then the **ReturnValue** is **0**.

The screenshot shows the 'Callout Functions' dialog box. The 'Callout Functions' tab is selected. In the left sidebar, there is a tree view of callout types, with 'Common callouts' expanded and 'Data Source Callouts' selected. Under 'Data Source Callouts', two items are listed: 'Execute a data source' and 'Refresh data from a data source'. The 'Execute a data source' item has a red border around it, indicating it is the active or selected function. The 'Remarks' column provides detailed descriptions for each function.

Callout function	Syntax	Remarks
Execute a data source	(DataSourceExecute, "data source name", "Key Param1")	Executes the data source using the Key and returns the data. A value of 0 will be returned if the key is invalid.
Refresh data from a data source	(DataSourceRefresh, "data source name", "Key Param1")	Executes the data source using the Key and refreshes the fields associated with the data source. A value of 0 will be returned if the key is invalid.

Let's see this working:

In our application, we can add a button labelled **Execute Data Source**, whose control ID is 1001.



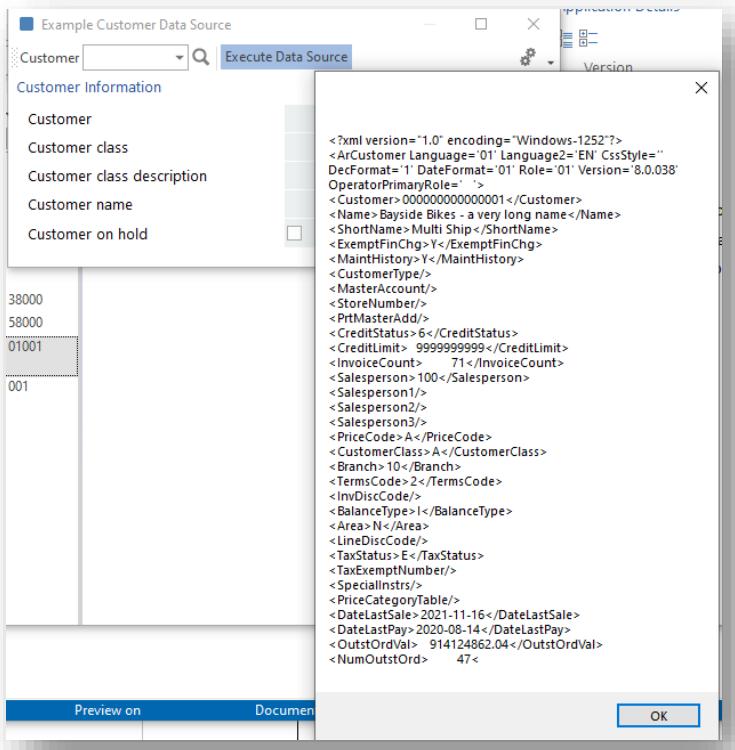
Next, in the script, we can add this code:

```
Function AppDesigner_OnUserEvent(EventType, EventName, EventID, Param1, Param2, Param3)

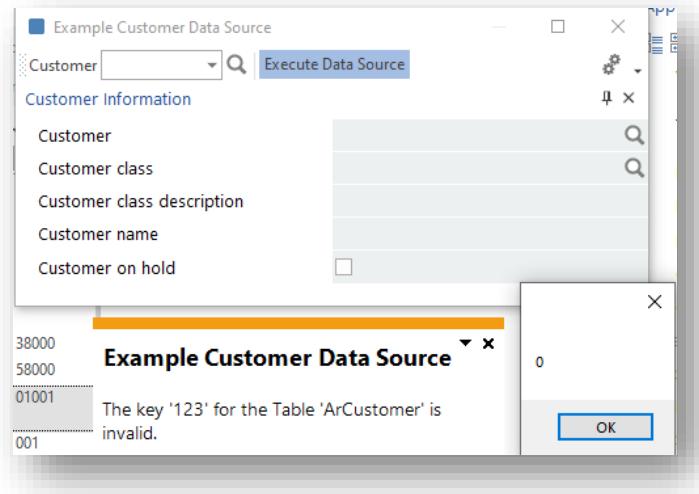
if EventType = ToolbarEvent then
    if EventID = 1001 then
        returnValue = CallSYSPROFunction(DataSourceExecute, "Customer Master",
"1|Param1")
        msgbox ReturnValue
    end if
end if

End Function
```

When clicking on the button, the data source **Customer Master** will be executed with a KEY being passed of '1' and a message box will be displayed with the contents of the data returned from the data source:



If we change the script to pass an invalid key, this is what happens:



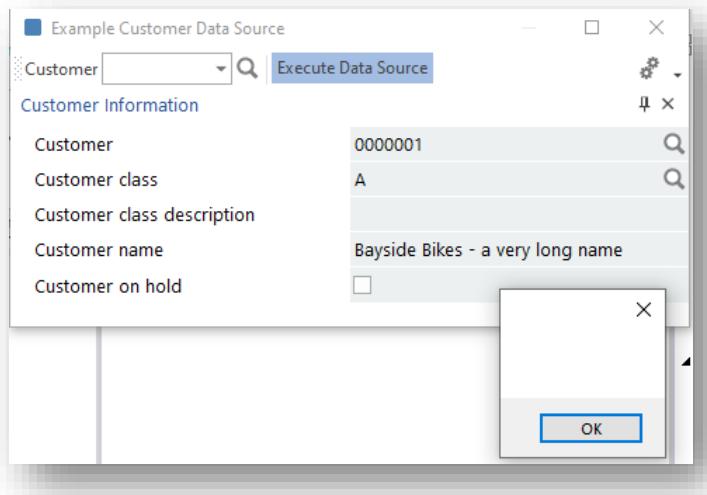
A toast notification is given indicating the invalid key, and the return value from the data source is '0'.

It's at this point you may wish to change the **Show toast notification if invalid key** option against the data source to be unchecked – this is useful if the script is to decide what action to take for an invalid key instead.

Now let's change the callout to be an automatic refresh of form fields:

```
if EventType = ToolbarEvent then
    if EventID = 1001 then
        returnValue = CallSYSPROFunction(DataSourceRefresh, "Customer Master",
"1|Param1")
        end if
    end if
End Function
```

And when we click on the button, this is what happens:



The form fields are now refreshed, and the return value is blank indicating the callout was successful.

DATA SOURCE EXECUTION USING COMPLEX SQL STATEMENTS OR XML DOCUMENTS

You can execute two script callouts for data sources where the SQL statement or XML document for a business object transaction are too complex to define within the data source form. In both cases, the callouts return the data from the executed data source, or a value of '0' if the data source cannot be executed successfully.

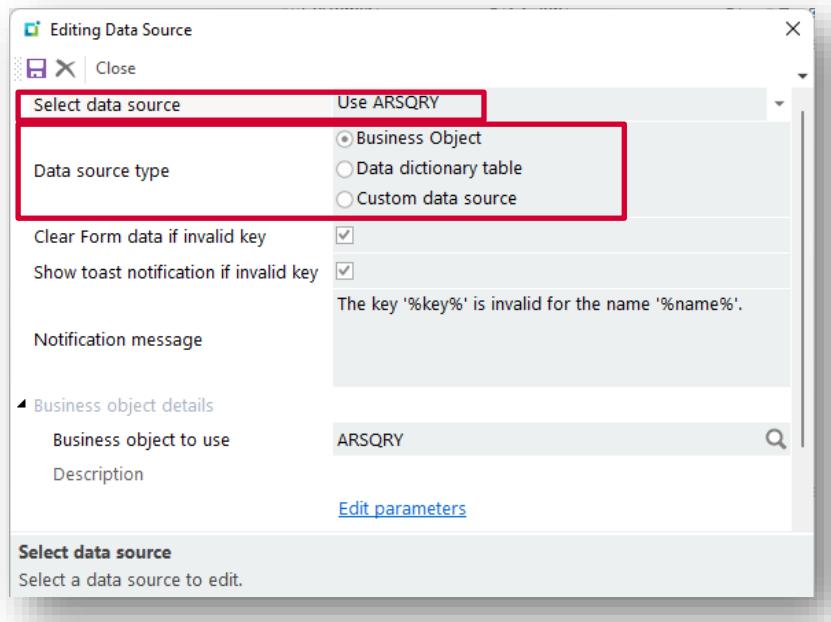
Execute a data source (DataSourceExecuteSQL, "data with SQL statement source name", SQLStatement)	Executes the data source using the supplied SQL statement. The data source must be for a transactional custom type.
Execute a data source (DataSourceExecuteXMLDoc, with document "data source name", xmlDoc)	Executes the data source using the supplied XML document string. The data source must be for a transactional Business Object type.

The callout **DataSourceExecuteSQL** allows you to execute a custom data source and supply the entire SQL statement. This can be used if the SQL statement that's embedded in the data source cannot be sufficiently described, or if you need to generate a SQL statement during the execution of the script.

The callout **DataSourceExecuteXMLDoc** allows you to execute a transactional business object data source and supply the complete XML document. This can be useful if the XML document defined in the data source mapping is not sufficient to fully describe the document, or if you need to generate the XML document during the execution of the script.

USING A BUSINESS OBJECT AS A DATA SOURCE

A SYSPRO business object provides more information that can be used as a data source than a table. In the following example, a data source name of **Use ARSQRY** has been defined and the **Business Object** option has been selected for the data source type.



To use a SYSPRO business object as a data source, proceed as follows:

1. Supply the name of the business object.

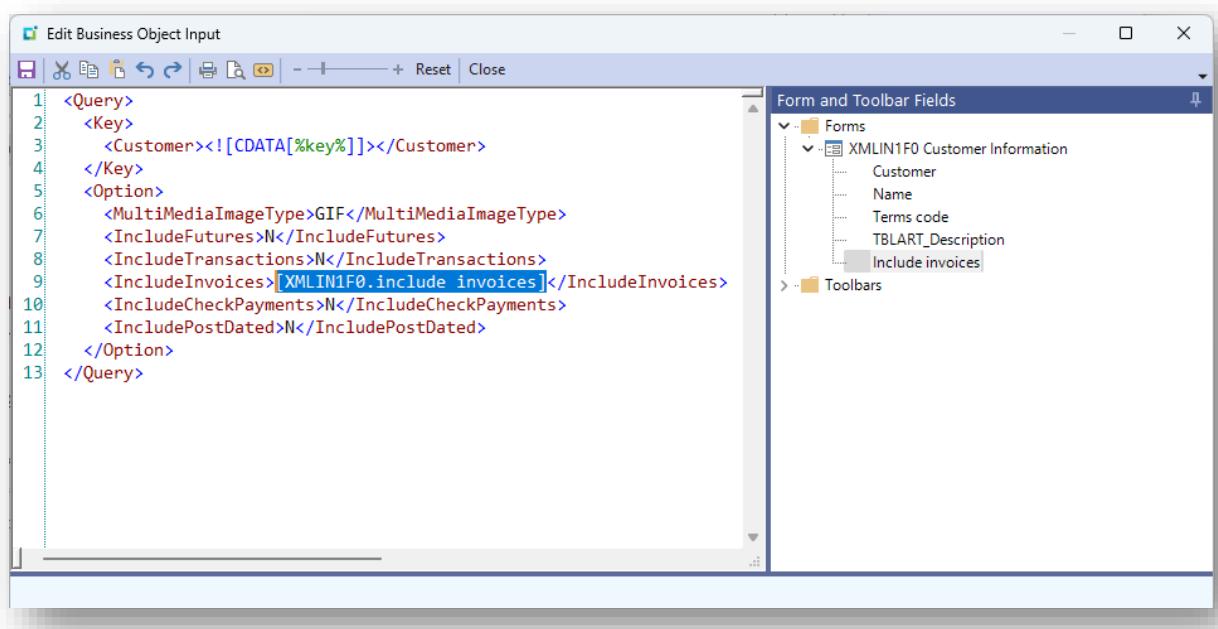
Considerations:

- This must be of type Query.
- After entering a business object, a default set of input parameters is created based on the XML derived from the ...\\base\\Schemas folder using the business object name (e.g. ...\\base\\schemas\\ARSQRY.XML).

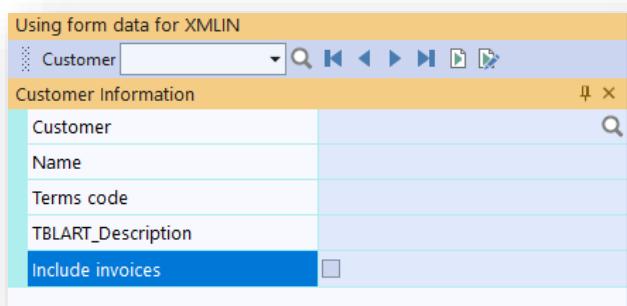
2. Edit the parameters to be supplied for the business object by clicking on **Edit parameters**.

An editor will be displayed with the default XMLIN parameters, including the placeholder **%key%** for the key to be supplied.

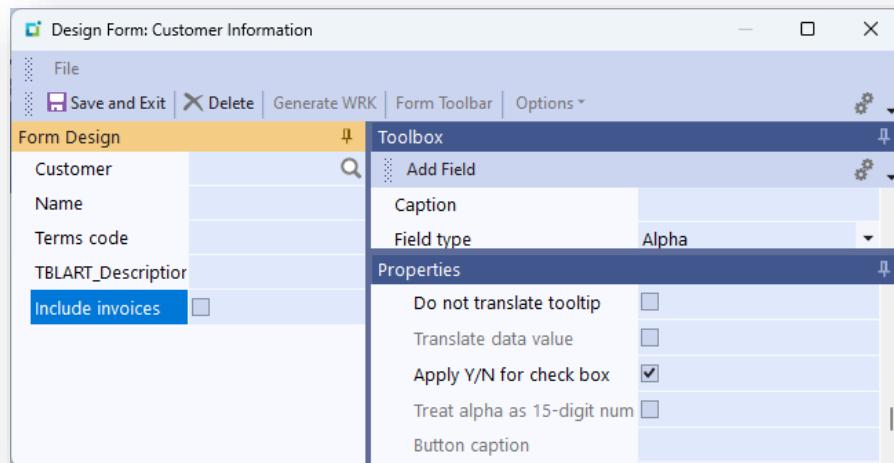
You can insert a placeholder for form and toolbar fields into the XMLIn statement for a business object. This is useful if you wish the XMLIn statement to be dynamically altered at run time by the value of one or more form or toolbar fields.



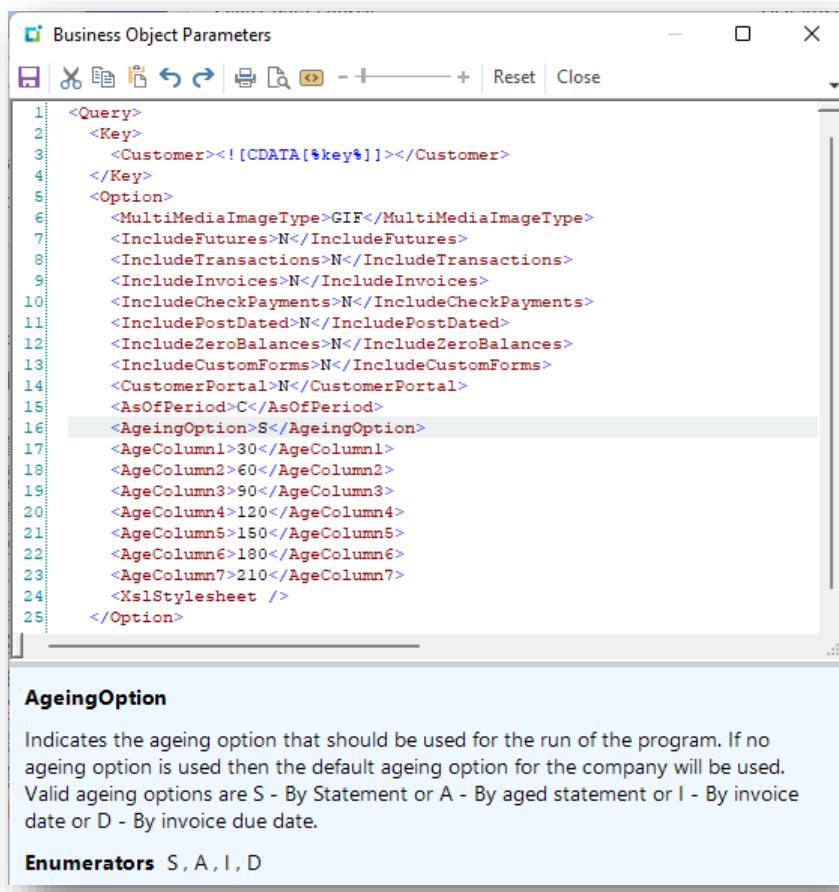
In this example, the value of the form field **Include invoices** from the form **XMLIN1FO** will be inserted into the XMLIn string for the business object at runtime.



Note that the XMLIn parameter for the business object (in this case, ARSQRY) is expecting a value of Y or N, so it's necessary to tick the option **Apply Y/N for check box** against the form field **Include invoices**, as indicated below:



- Click the **SAVE** button to save any changed parameters or click **Reset** to reset the XMLIN back to the original values.



4. If you intend to use the XML output from the business object in script rather than binding data to form fields then select the option **Return all elements** – the complete document will be returned rather than a subset determined by the single XML node which is selected by clicking **Select XML node**.

When you select a XML node, the available fields will be shown – these are the fields that can be selected and added to a form.

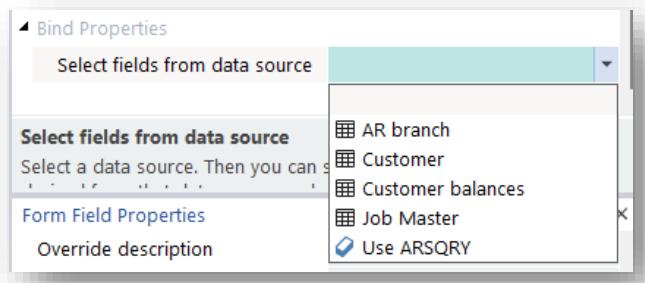
The screenshot shows the 'Business Object Properties' dialog. On the left, the 'Select Primary Node' tree is displayed under 'XSD element'. A node named 'Header' is selected. On the right, a grid titled 'Available Fields' lists various XML elements and their properties:

Selected	XML element	Column caption	Data type
<input type="checkbox"/>	CompanyName	Company name	Default
<input type="checkbox"/>	CompAddress1	Comp address1	Default
<input type="checkbox"/>	CompAddress2	Comp address2	Default
<input type="checkbox"/>	CompAddress3	Comp address3	Default
<input type="checkbox"/>	CompTaxNo	Comp tax no	Default
<input type="checkbox"/>	CompRegNo	Comp reg no	Default
<input type="checkbox"/>	Customer	Customer	Default
<input type="checkbox"/>	Name	Name	Default
<input type="checkbox"/>	SoldToAddr1	Sold to addr1	Default
<input type="checkbox"/>	SoldToAddr2	Sold to addr2	Default
<input type="checkbox"/>	SoldToAddr3	Sold to addr3	Default
<input type="checkbox"/>	SoldToAddr3Loc	Sold to addr3 loc	Default
<input type="checkbox"/>	SoldToAddr4	Sold to addr4	Default
<input type="checkbox"/>	SoldToAddr5	Sold to addr5	Default

5. Once the data source has been saved, you can select the fields to be shown on a form. Here, a blank form has been created:

The screenshot shows the 'Application Designer: Data Source Example (DTS001)' window. The 'Toolbox' on the left contains various controls like Chart, Flowgraph, Form, etc. The main area shows a form with fields 'Customer' and 'Job'. Below it is a 'Data Source Explorer' pane. A 'Form Details - ARSQRY data source fields' pane on the right lists properties such as 'OnAfterSubmit event supported', 'Disallow field properties', and 'Select fields from data source'. A 'Select fields from data source' section allows selecting fields from a data source. A 'Form Field Properties' section at the bottom right includes 'Caption', 'Override description', 'Type', 'Change field type', and 'Length' set to 0.

6. From the **Select fields from data source** list, select **Use ARSqry**:



The fields derived from the **Header** XML node in the business object ARSqry are displayed:

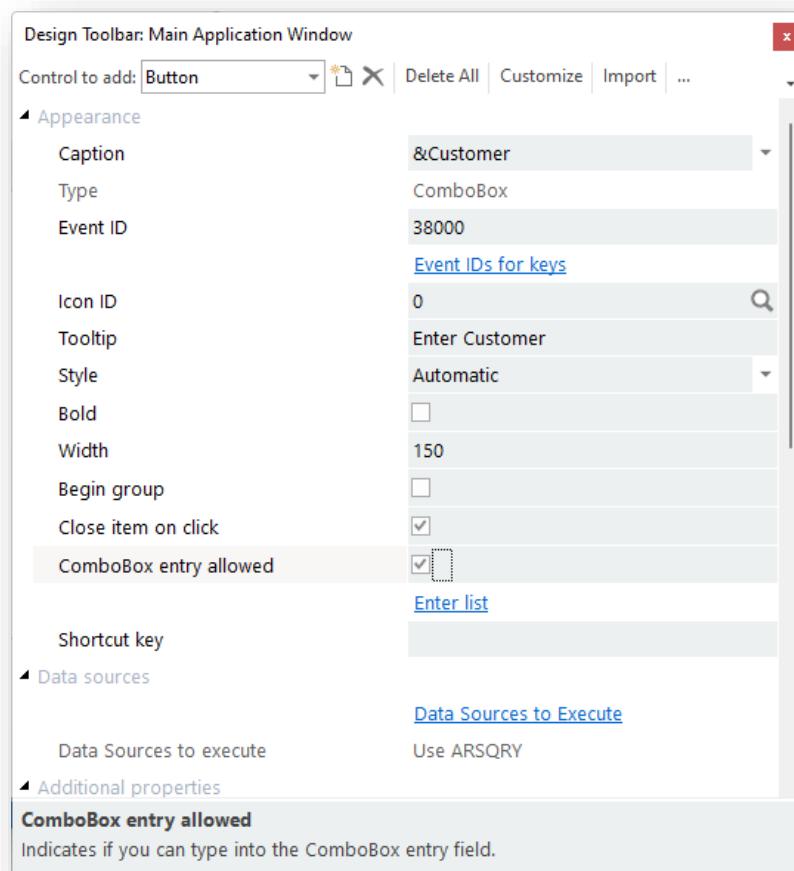
Select	Description	Data type	Column name	Len...	Deci...	Field type
<input type="checkbox"/>	Currency	Alpha	Currency	0		
<input type="checkbox"/>	TBLCUR_Description	Alpha	TBLCUR_Descript...	0		
<input type="checkbox"/>	Contact	Alpha	Contact	0		
<input type="checkbox"/>	Credit limit	Alpha	CreditLimit	0		
<input type="checkbox"/>	Edited_CreditLimit	Alpha	Edited_CreditLimit	0		
<input type="checkbox"/>	Over limit	Alpha	OverLimit	0		
<input type="checkbox"/>	Translated_OverLimit	Alpha	Translated_Over...	0		
<input type="checkbox"/>	Credit check flag	Alpha	CreditCheckFlag	0		
<input type="checkbox"/>	Short name	Alpha	ShortName	0		
<input type="checkbox"/>	Customer type	Alpha	CustomerType	0		
<input type="checkbox"/>	Customer type desc	Alpha	CustomerTypeD...	0		
<input type="checkbox"/>	Master account	Alpha	MasterAccount	0		
<input type="checkbox"/>	Store number	Alpha	StoreNumber	0		
<input type="checkbox"/>	Credit status	Alpha	CreditStatus	0		
<input type="checkbox"/>	Credit status descri...	Alpha	CreditStatusDes...	0		
<input checked="" type="checkbox"/>	Invoice count	Alpha	InvoiceCount	0		
<input type="checkbox"/>	Salesperson	Alpha	Salesperson	0		
<input type="checkbox"/>	Customer class	Alpha	CustomerClass	0		
<input type="checkbox"/>	Customer class desc	Alpha	CustomerClassD...	0		
<input type="checkbox"/>	Inv disc code	Alpha	InvDiscCode	0		
<input type="checkbox"/>	Line disc code	Alpha	LineDiscCode	0		
<input type="checkbox"/>	Balance type	Alpha	BalanceType	0		

7. Select the fields to be included in the form and click **Add Selected Items**.

The selected fields will be displayed on the form:

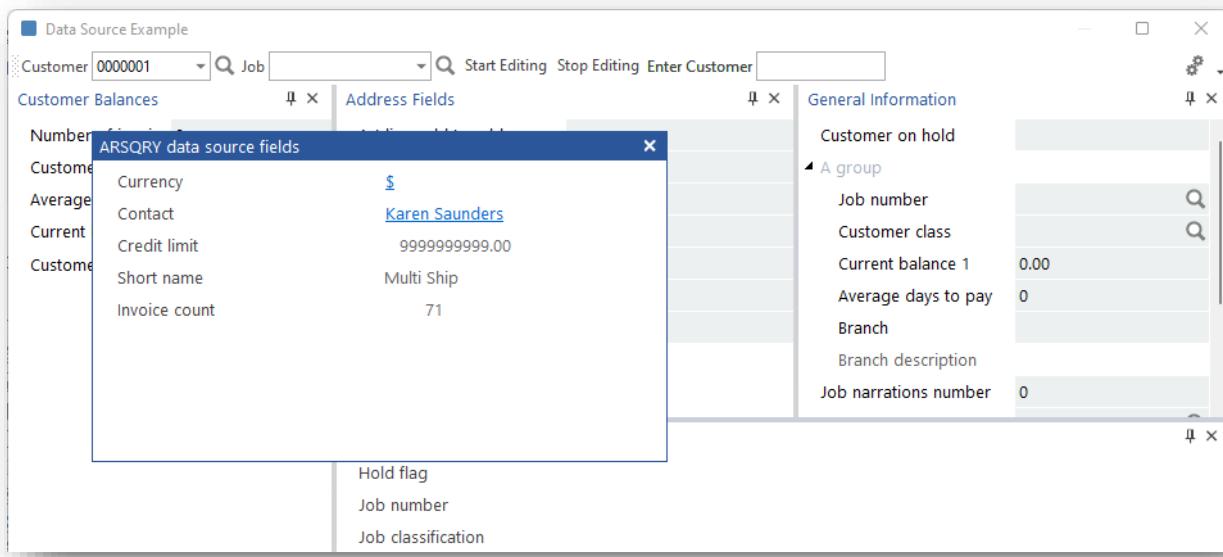
The screenshot shows the Application Designer interface with the 'Data Source Example' project open. The 'Address Fields' tab is selected in the main workspace. A list box titled 'ARSqry data source fields' contains several items, with 'Credit limit' currently selected. To the right, the 'Form Details - ARSqry data source fields' panel shows field properties for 'Credit limit', including its caption, data type (Alpha), length (0), and field binding ([Use ARSqry]CreditLimit). The 'Form Field Properties' panel also displays these settings.

8. Finally, against a toolbar labelled **Customer** we can select the data source **Use ARSQRY** to be executed:



You can now run the application.

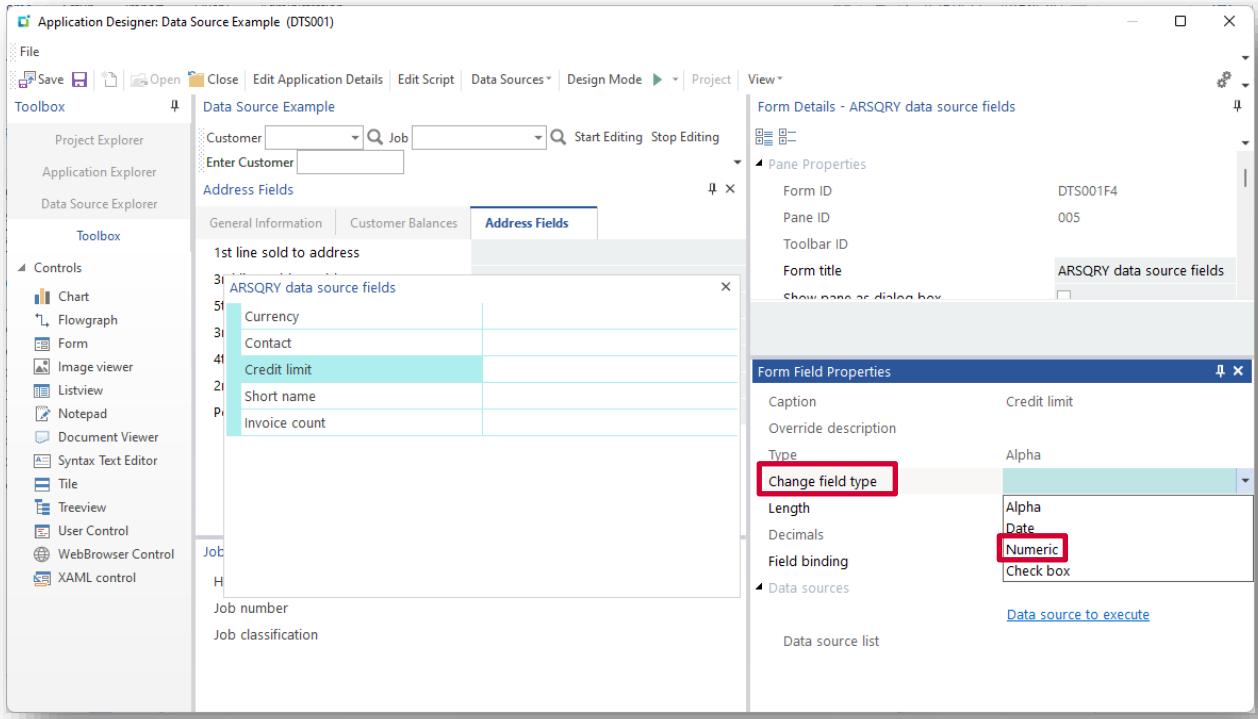
After entering a valid Customer number, you will see the form fields populated with data. However, it may be noticed that all fields have been treated as type **Alpha**:



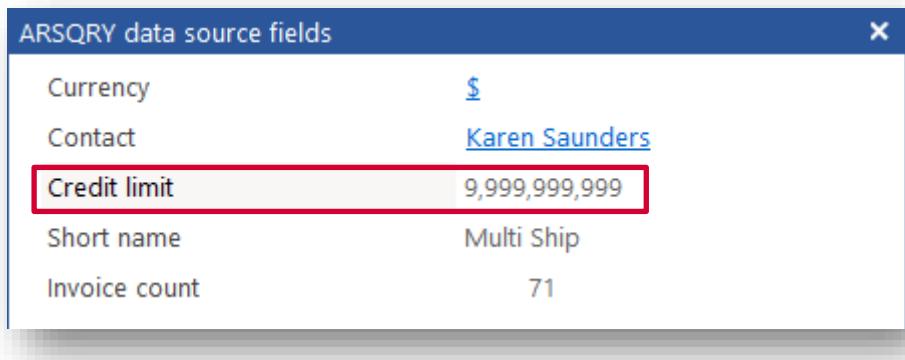
This is because the fields derived from a business object are not type-classed.

You can change the field type in the Designer:

1. Click on **Credit limit** and the Field Properties will be updated.
2. Select **Numeric** from the **Change field type** list:



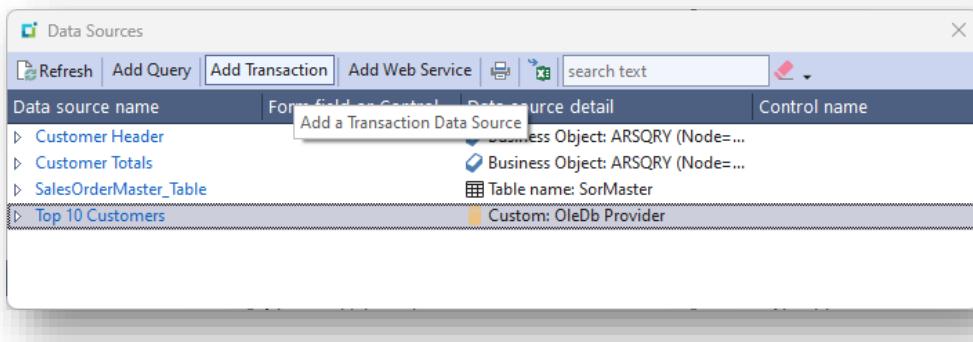
3. Enter a length of 10 and decimals of 0, and then run the application again.
This time the Credit Limit is shown without decimals and with thousand separators:



You can change any form field in a similar manner, as long as the field is derived from a business object. If you know that a field can contain only a single character value of either Y or N, then you can use the field type of **Check box**.

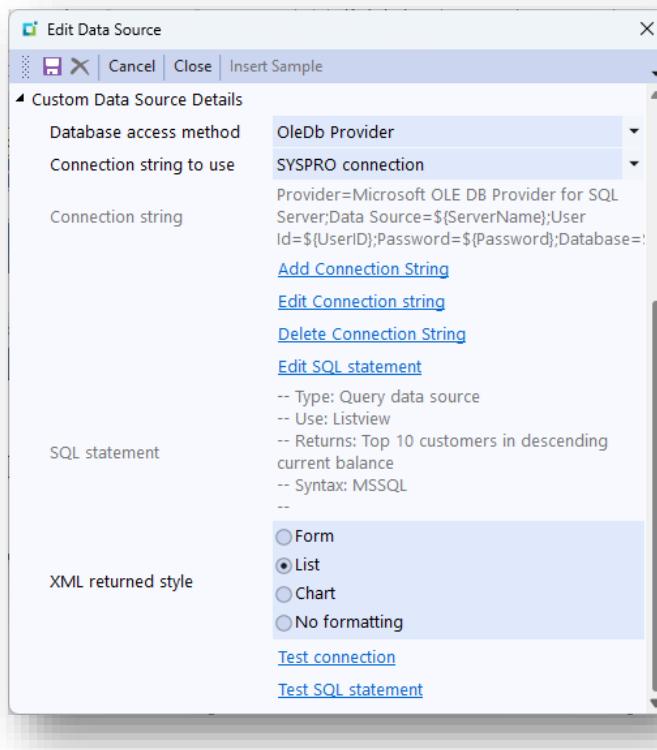
CUSTOM DATA SOURCES

You can create Custom Data Sources using direct SQL statements and Connection Strings. You will need to decide if the data source to be added is a Query or Transaction from the data source window:



A transaction data source can be executed just like a Query data source. Additionally, there is a special script callout **DataSourceExecuteSQL** that may be used if the SQL statement is too complex to construct from within the editor:

Execute a data source with SQL statement (DataSourceExecuteSQL, "data source name", SQLStatement) Executes the custom data source using the supplied SQL statement and returns the data.



You must select the database access method and then define a connection string and SQL statement.

You can also select T-SQL as a database access method. T-SQL or Transact SQL is the query language specific to the Microsoft SQL Server product. It can help perform operations like retrieving the data from a single row, inserting new rows, and retrieving multiple rows. Syntax such as using FOR XML requires using T-SQL syntax. It also enables the developer to call stored procedures in the MSSQL database and return formatted XML.

You can execute SQL statements for any type of database, not just a SYSPRO database.

The XML returned from a custom data source can be 'unformatted' using the **No formatting** style. This is useful if the SQL statement includes the FOR XML syntax using a T-SQL syntax as SQL Server can then return the XML as it seems appropriate. It is also useful if you are calling SQL stored procedures that format XML that you want to return to your script.

If you wish the output from the SQL statement to be used for a form, listview or chart then select the appropriate style – XML in a pre-determined format will be returned no matter which SQL statement is issued.

XML RETURNED STYLE FOR A FORM

The XML is returned in a format suitable for populating a form:

```
<Form><Fields>
  <Field Caption="StockCode" Description="Stock code" Value="A100"
Type="Alpha" Length="30" />
  <Field Caption="Description" Description="Description" Value="15 Speed
Mountain Bike Boys" Type="Alpha" Length="50" />
</Fields></Form>
```

XML RETURNED STYLE FOR A LIST VIEW

The XML is returned in a format suitable for populating a listview with iterating rows:

```
<Query>
<Row>
  <StockCode>A100</StockCode>
  <Description>15 Speed Mountain Bike Boys</Description>
  <Warehouse>E</Warehouse>
  <QtyOnHand>950.000000</QtyOnHand>
  <DateLastSale>20150412</DateLastSale>
</Row>
<Row>
  <StockCode>A100</StockCode>
  <Description>15 Speed Mountain Bike Boys</Description>
  <Warehouse>FG</Warehouse>
  <QtyOnHand>6.000000</QtyOnHand>
  <DateLastSale>
  </DateLastSale>
</Row>
<Row>
</Query>
```

XML RETURNED STYLE FOR A CHART

The XML is returned in a format suitable for populating a Chart with multiple chart points:

```
<Points>
<Point Label='E' Value ='950.000000'/>
<Point Label='FG' Value ='6.000000'/>
```

```

<Point Label='N' Value ='909.000000'/>
<Point Label='S' Value ='430.000000'/>
</Points>

```

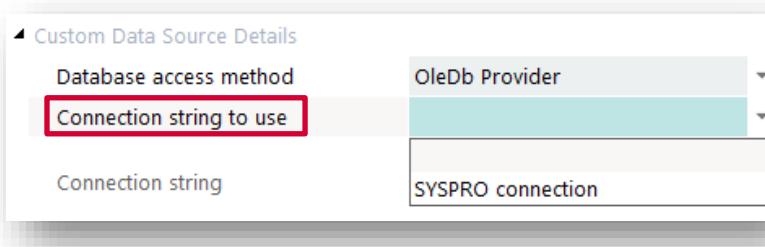
CONNECTION STRING

The connection string serves to:

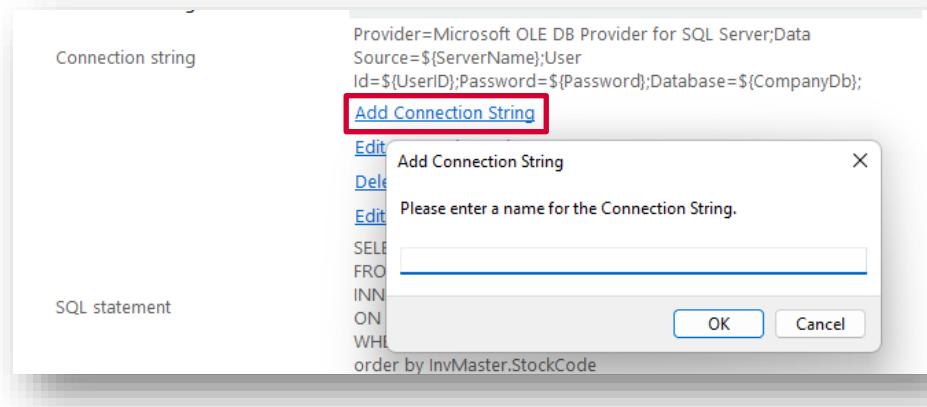
- Identify the server instance and database to connect to
- Determine what driver, login, etc. to use to connect to the SQL Server instance.

In an application, you can have multiple connection strings to target different databases, and each connection string is given a 30-character name.

For each data source you can select which connection string you wish to use from the **Connection String** drop down list:

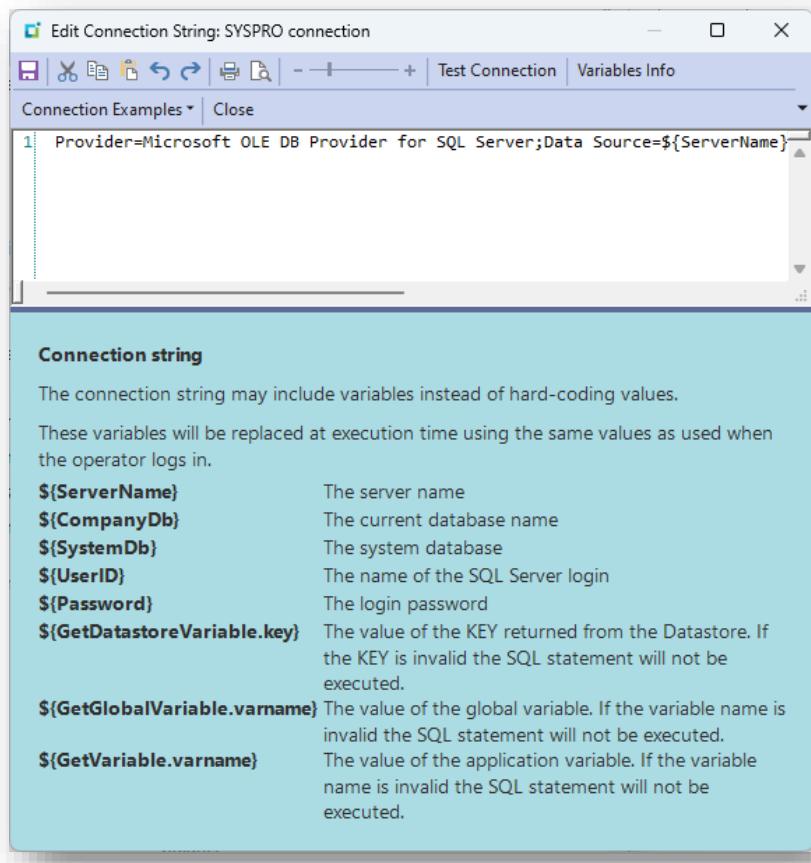


By default, a **SYSPRO connection** string is always supplied. This connection will connect to the same SYSPRO SQL Server instance as used for the operator that's logged into SYSPRO. Click **Add Connection String** to add a new connection string. You will be prompted to enter a name. The name must be unique within the application.

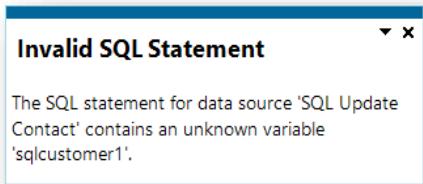


You can now edit the connection string. There are some sample connection strings that you can insert and then adjust as necessary.

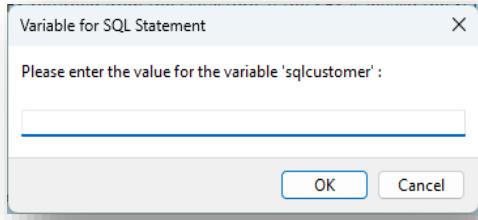
Note that you can inject global variables, application script variables, datastore keys and other values into connection strings and SQL statements using placeholders so that the connection string can be dynamically constructed as the application executes a data source:



If a variable cannot be found when executing the SQL statement, then a message like this will be shown:



In addition, when editing a data source you can test both a connection string and a SQL statement. If either (or both) statements contain variables, then when you click 'Test..' you will be prompted to enter the variable(s).



Click **Edit SQL statement** to enter a SQL statement.

The tree view in the Database Information pane provides all the special variables, database tables, database views, form field values and toolbar control values that are available to be

used in the SQL statement based on the Connection String. Double-click on any item to insert it into the SQL statement – these placeholders will be updated with appropriate values when the SQL statement is executed.

The screenshot shows the 'Edit SQL statement' dialog box. On the left, a code editor displays a sample SQL query:

```

1 -- Type: Query data source
2 -- Use: Listview
3 -- Returns: Top 10 customers in descending current balance
4 -- Syntax: MSSQL
5 --
6 SELECT TOP 10 ArCustomer.Customer, ArCustomer.Name, ArCustomer.CreditLimit, ArCustomer.Salesperson,
7 ArCustomer.Area, ArCustomer.DateLastPay, ArCustomer.OutstandingBalance, ArCustomer.Contact, ArCustomer.Telephone, ArCustomerBal.CurrentBalance
8
9

```

The right side of the dialog has a sidebar titled 'Database Information' with sections for Variables, Tables, Views, Forms, and Toolbars. Below the sidebar, a section titled 'The SQL statement' lists various variables and their descriptions:

%key%	The placeholder for the primary key
%keyn%	Extra key placeholders (up to 8), numbered %key2% upwards.
 \${Company}	Company ID (e.g. EDU1)
 \${CompanyDb}	Current company database
 \${LocalCurrency}	Local currency code
 \${SharedInventoryDb}	Shared Inventory database
 \${SharedGIDb}	Shared General Ledger database
 \${SystemDb}	System database
 \${today}	Today's date in CCYY-MM-DD format
 \${GetDatastoreVariable.key}	The value of a KEY returned from the Datastore. If the KEY is invalid the SQL statement will not be executed.
 \${GetGlobalVariable.varname}	The value of the global variable. If the variable name is invalid the SQL statement will not be executed.
 \${GetVariable.varname}	The value of the application variable. If the variable name is invalid the SQL statement will not be executed.
 [ControlName.Caption]	A placeholder for the value derived from a form or toolbar field at execution time.

At the bottom of the dialog, there is an example SQL statement:

Example: `SELECT InvMaster.* , FROM InvMaster WHERE InvMaster.StockCode='%key%' and`

Considerations:

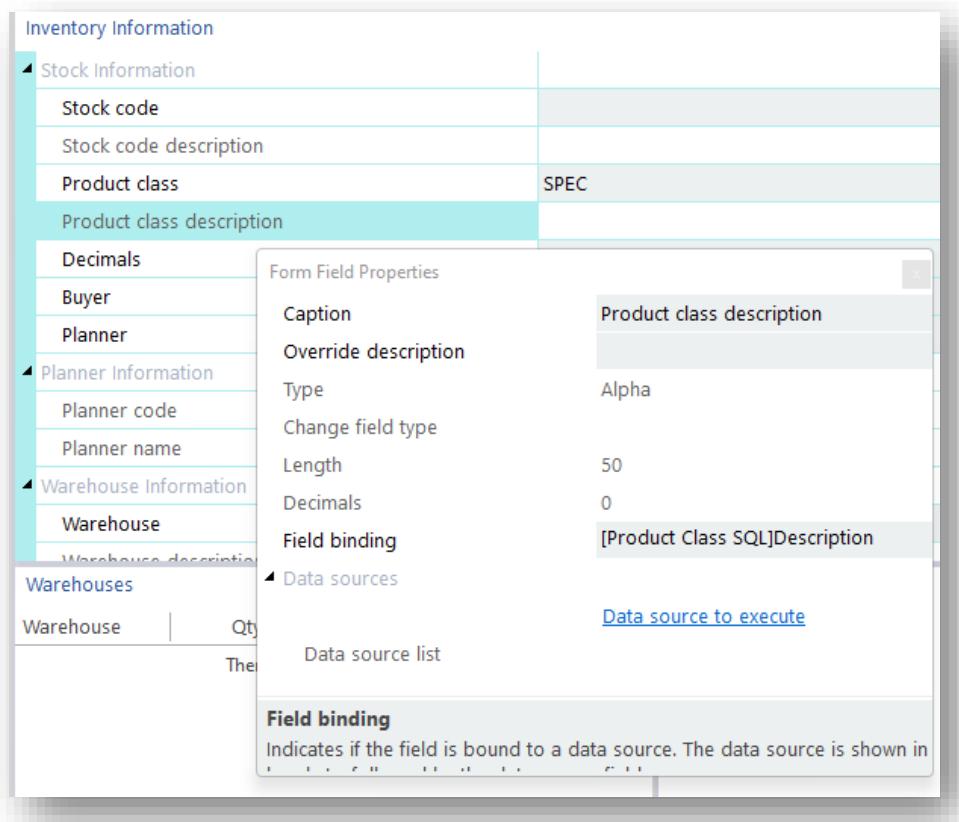
- You can use the variable **%key%** which will be replaced at execution time by the KEY that was used to execute the data source. You can also supply up to 9 keys to be used in the execution of a data source. The keys are supplied separated by a comma and will replace the placeholders %key%, %key2%, %key3% and so on. This is especially useful for complex SQL statements requiring multiple keys to be entered.
- Note:** A key cannot exceed 256 characters.
- There are other variables that may be used in the SQL statement rather than hard coding for databases or company names.
 - There are also two extra Keys (**%key2%** and **%key3%**) that may be used when the data source is to be executed from a Chart drill down.

- You can also use a variable to denote the value of a form field or editable toolbar control that will be inserted into the SQL statement at execution time. The syntax for this is **[formname.field caption]**, or **[toolbarName.EventID]**.
- You can view the available fields for all forms and toolbars in the application by expanding the **Forms/Toolbars** item in the tree view. Selecting a form field or toolbar control will inject the correct variable value into the SQL statement.

For example, this SQL statement:

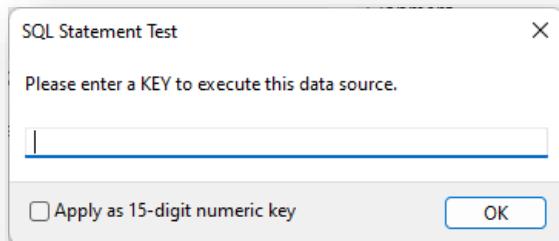
```
SELECT SalProductClassDes.* FROM SalProductClassDes where
SalProductClassDes.ProductClass = '[OLEDB1F0.Product class]'
```

will be executed after the current value of the form field **Product class** has been extracted from the form name **OLEDB1F0** in the application:

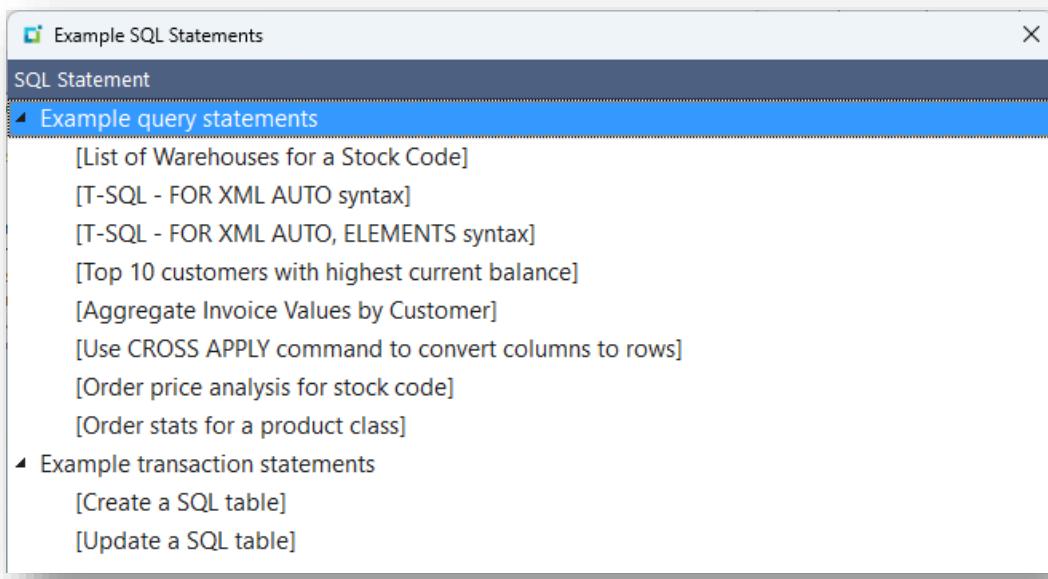


When you have constructed the SQL statement you can test it by clicking on the **Test SQL Statement**. If the statement contains any **%key%** placeholders, then you will be prompted to enter a KEY so that the statement can execute correctly.

If you know that the KEY is a numeric type (for example, for a CUSTOMER numeric key) then you can check the **Apply as 15-digit numeric key** option to save having to enter the full 15-digit number:

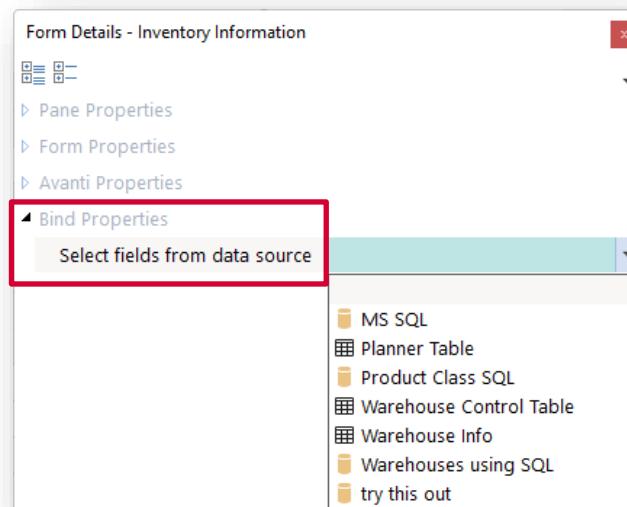


Numerous examples of SQL statements can be viewed and selected for use. Click on **SQL Examples** and then double-click on any statement for it to be inserted into the SQL editor.



ADDING CUSTOM DATA SOURCE FIELDS TO A FORM

Select the data source you want to use from the drop down list in the **Bind Properties**:



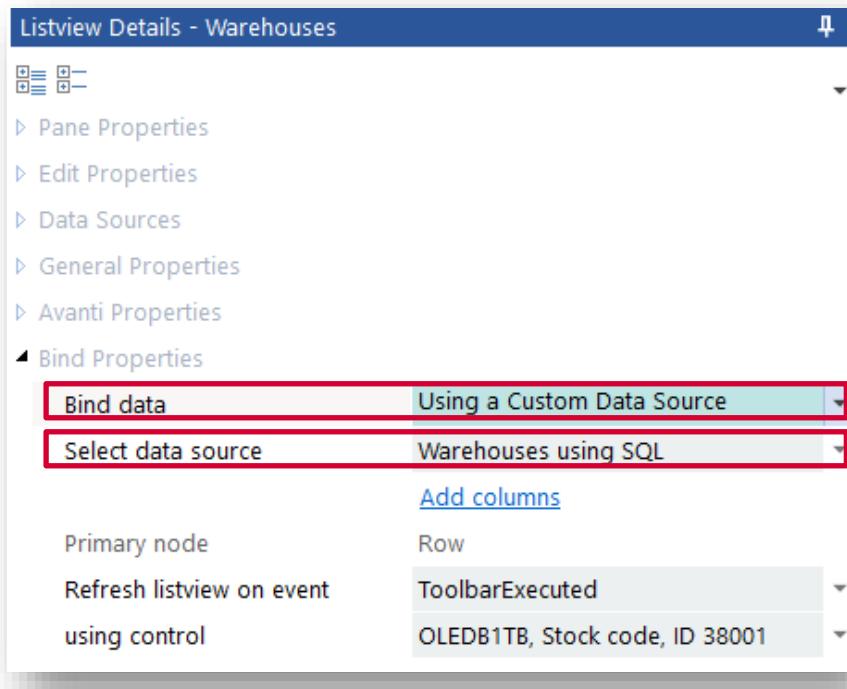
The available fields that are derived from the SQL statement will be shown:

Select	Description	Data type	Column name	Length	De...	Field ty...
<input type="checkbox"/>	Stock code	Alpha	StockCode	30		
<input type="checkbox"/>	Description	Alpha	Description	50		
<input type="checkbox"/>	Long desc	Alpha	LongDesc	100		
<input type="checkbox"/>	Alternate key1	Alpha	AlternateKey1	20		
<input type="checkbox"/>	Alternate key2	Alpha	AlternateKey2	20		
<input type="checkbox"/>	Ecc user	Alpha	EccUser	20		
<input type="checkbox"/>	Stock uom	Alpha	StockUom	10		
<input type="checkbox"/>	Alternate uom	Alpha	AlternateUom	10		
<input type="checkbox"/>	Other uom	Alpha	OtherUom	10		
<input type="checkbox"/>	Conv fact alt uom	Numeric	ConvFactAltUom	12	6	
<input type="checkbox"/>	Conv mul div	Alpha	ConvMulDiv	1		
<input type="checkbox"/>	Conv fact oth uom	Numeric	ConvFactOthUom	12	6	
<input type="checkbox"/>	Mul div	Alpha	MulDiv	1		
<input type="checkbox"/>	Mass	Numeric	Mass	18	6	
<input type="checkbox"/>	Volume	Numeric	Volume	18	6	
<input type="checkbox"/>	Decimals	Numeric	Decimals	1	0	
<input type="checkbox"/>	Price category	Alpha	PriceCategory	1		
<input type="checkbox"/>	Price method	Alpha	PriceMethod	1		
<input type="checkbox"/>	Supplier	Alpha	Supplier	15		

Select the required fields and click on **Add Selected Items**.

ADDING CUSTOM DATA SOURCE COLUMNS TO A LISTVIEW

Against the listview properties, select the option **Using a Custom Data Source** from the **Bind data** option. Then select the data source from the drop down list.



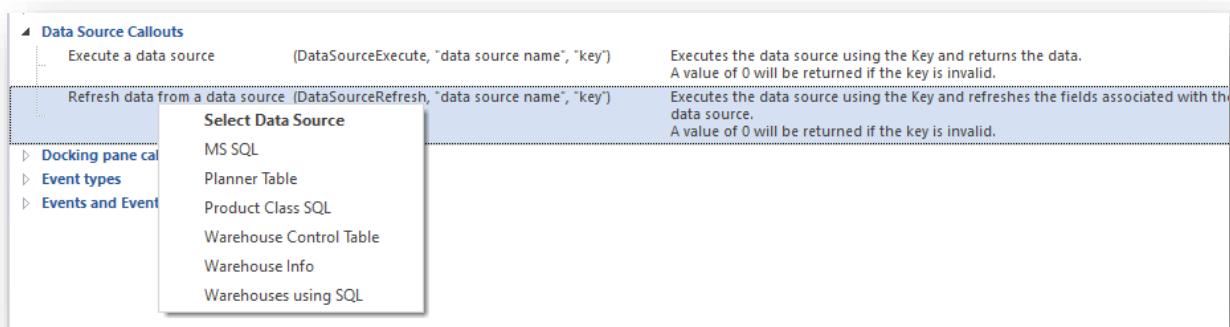
Next, click on **Add columns**. This will show all the columns that are available to be shown in the listview based on the SQL statement:

Select	Description	Data type	Column name	Length	De...	Field ty...
<input type="checkbox"/>	Stock code	Alpha	StockCode	30		
<input type="checkbox"/>	Stock code	Alpha	StockCode	30		
	Warehouse	Alpha	Warehouse	10		
	Qty on hand	Numeric	QtyOnHand	18	6	
<input type="checkbox"/>	Mtd qty received	Numeric	MtdQtyReceived	18	6	
<input type="checkbox"/>	Mtd qty adjusted	Numeric	MtdQtyAdjusted	18	6	
<input type="checkbox"/>	Mtd qty trf	Numeric	MtdQtyTrf	18	6	
<input type="checkbox"/>	Mtd qty issued	Numeric	MtdQtyIssued	18	6	
<input type="checkbox"/>	Mtd qty sold	Numeric	MtdQtySold	18	6	
<input type="checkbox"/>	Qty allocated	Numeric	QtyAllocated	18	6	
<input type="checkbox"/>	Qty on order	Numeric	QtyOnOrder	18	6	
<input type="checkbox"/>	Qty on back order	Numeric	QtyOnBackOrder	18	6	
<input type="checkbox"/>	Minimum qty	Numeric	MinimumQty	18	6	
<input type="checkbox"/>	Maximum qty	Numeric	MaximumQty	18	6	
<input type="checkbox"/>	Ytd qty sold	Numeric	YtdQtySold	18	6	
<input type="checkbox"/>	Prev year qty sold	Numeric	PrevYearQtySold	18	6	
<input type="checkbox"/>	Opening balance	Numeric	OpeningBalance	18	6	
<input type="checkbox"/>	Ytd usage value	Numeric	YtdUsageValue	14	2	
<input type="checkbox"/>	Ahc class	Alpha	AhcClass	1		

By default, the PRIMARY NODE will be defined as **Row** and cannot be changed – that's because the data source will always return XML data with <Row> elements.

When the custom data source is executed, the listview is automatically populated according to the columns defined for the listview.

The data source can be executed automatically using the **Refresh listview on event** option, or can be executed directly from the VBScript data source callouts:



TRANSACTIONAL DATA SOURCES

Transactional data sources allow you to define data sources that update the SYSPRO or any external database using either a SYSPRO business object or a SQL statement.

A transaction data source can be executed just like any other Query data source. Additionally, there is a special script callout **DataSourceExecuteSQL** that may be used if the SQL statement is too complex to construct from within the editor:

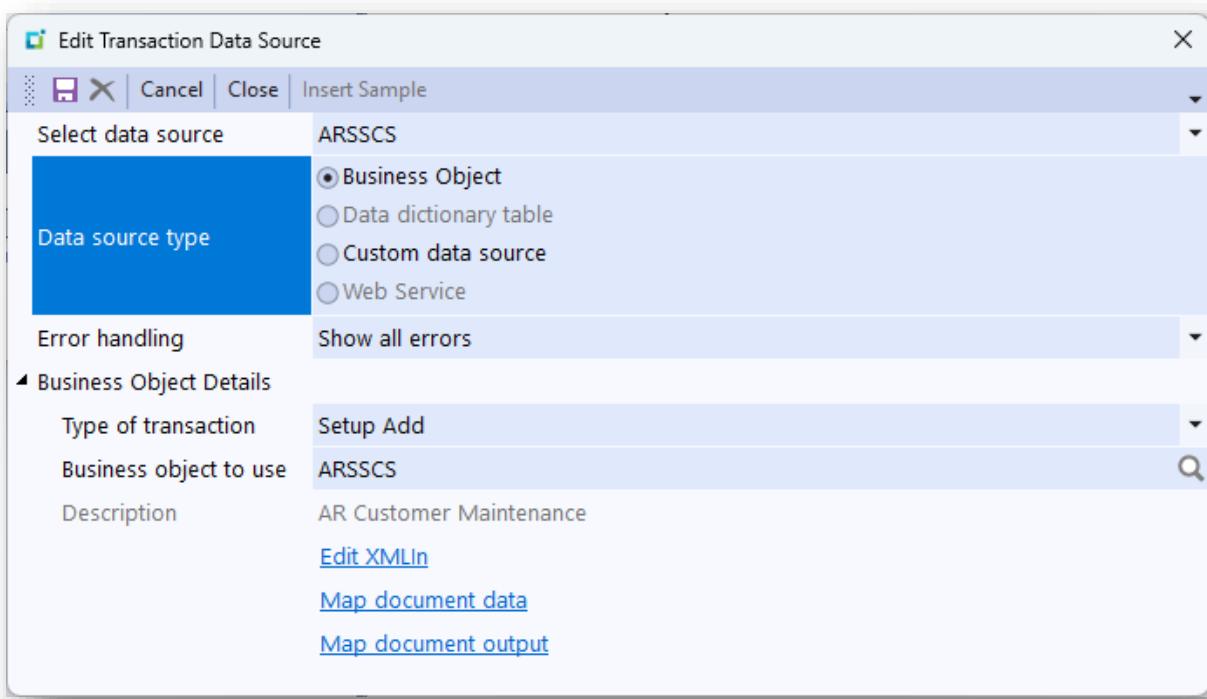
Execute a data source with SQL statement (DataSourceExecuteSQL, "data source name", SQLStatement) Executes the custom data source using the supplied SQL statement and returns the data.

DEFINING A TRANSACTIONAL BUSINESS OBJECT DATA SOURCE

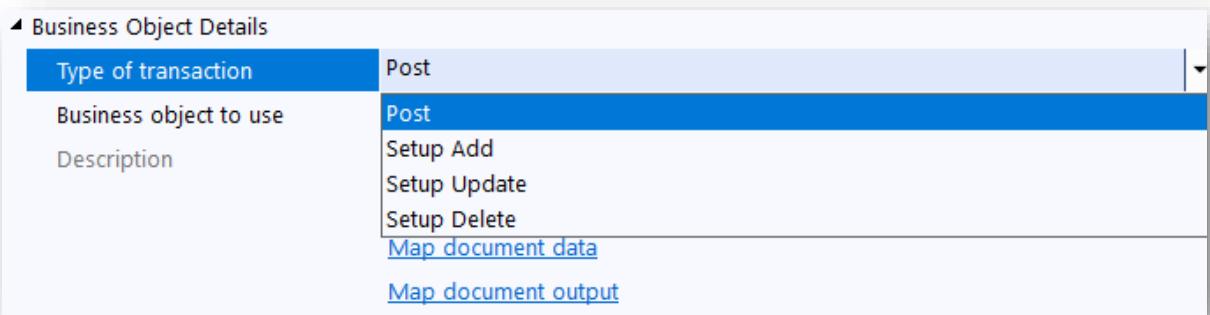
A transactional business object can be quite difficult to execute, often because of the complexity of creating the XML Document. Using a data source, you can alleviate some (if not all) of the effort of creating the XML document by mapping the XML elements required by the business object directly to form fields, toolbar fields and listview columns in your application. When executing the data source the XML document is then constructed exactly as required by the Business Object's XSD schema with the values derived from the mapped elements.

Here are the steps required to define this data source:

1. After entering a Data source name, select a data source type **Business Object**:



2. Next, select the type of transaction – Post, or one of the Setup types:



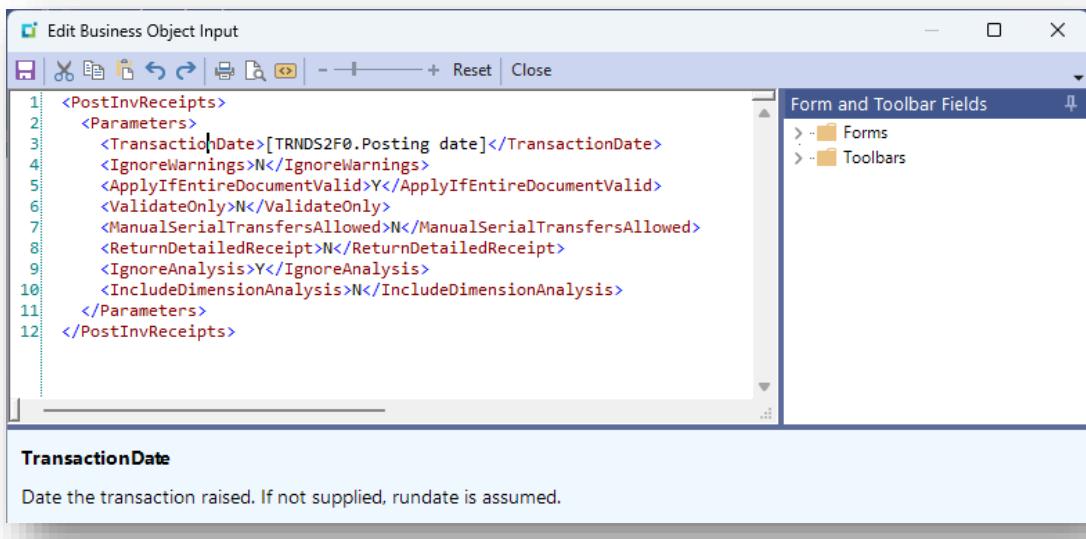
The **Setup** transaction types are reserved for business objects that update static data, such as a Customer or Stock code record. Such business objects can either ADD, UPDATE or DELETE records. An example is ARSSCS, the AR Customer Maintenance business object.

It is worth noting that you can change this Setup transaction type during the run of the application by using the callout **DataSourceSetupType**. So, if you select **Setup Add** and enter a business object name of ARSSCS, you can change this to be **Setup Update** using the script callout. This is important to be able to do this so that you only have a single mapped view of the document data.

Post transaction type is reserved for business objects that have business logic to execute transactions against the database. An example might be INVTMR, the Inventory Movements Receipts business object.

Note: Use the browse facility to search for the appropriate business objects for the type of transaction.

3. Now click on [Edit XMLIn](#) to edit the XML parameters for the business object. You will notice that you can inject placeholders into the XML content, such as form field values. You may also note that date form field values will be automatically transformed to be CCYY-MM-DD format, as required by business objects.



4. Click on [Map Document Data](#) to map the business object's XML elements to fields in your application.

Transaction business objects derive their data from a XML document that typically looks like this (the example is for ARSSCS, the Customer Setup Business Object):

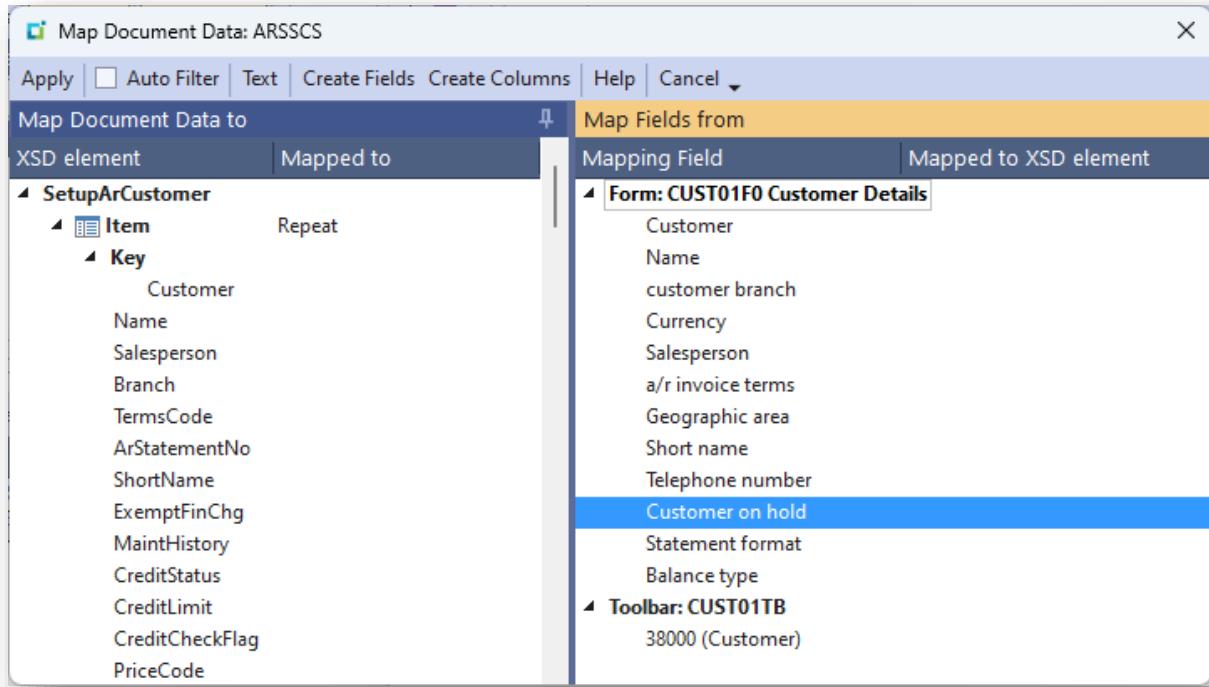
```
<SetupArCustomer>
  <Item>
    <Key>
      <Customer>000075</Customer>
    </Key>
    <Name>Johnson's Bicycles - South</Name>
    <Salesperson>100</Salesperson>
    ...
  </Item>
</SetupArCustomer>
```

The purpose of the mapping window is to simplify the process of creating the XML document by mapping XML elements to values derived from pre-defined form fields, toolbar fields or listview columns.

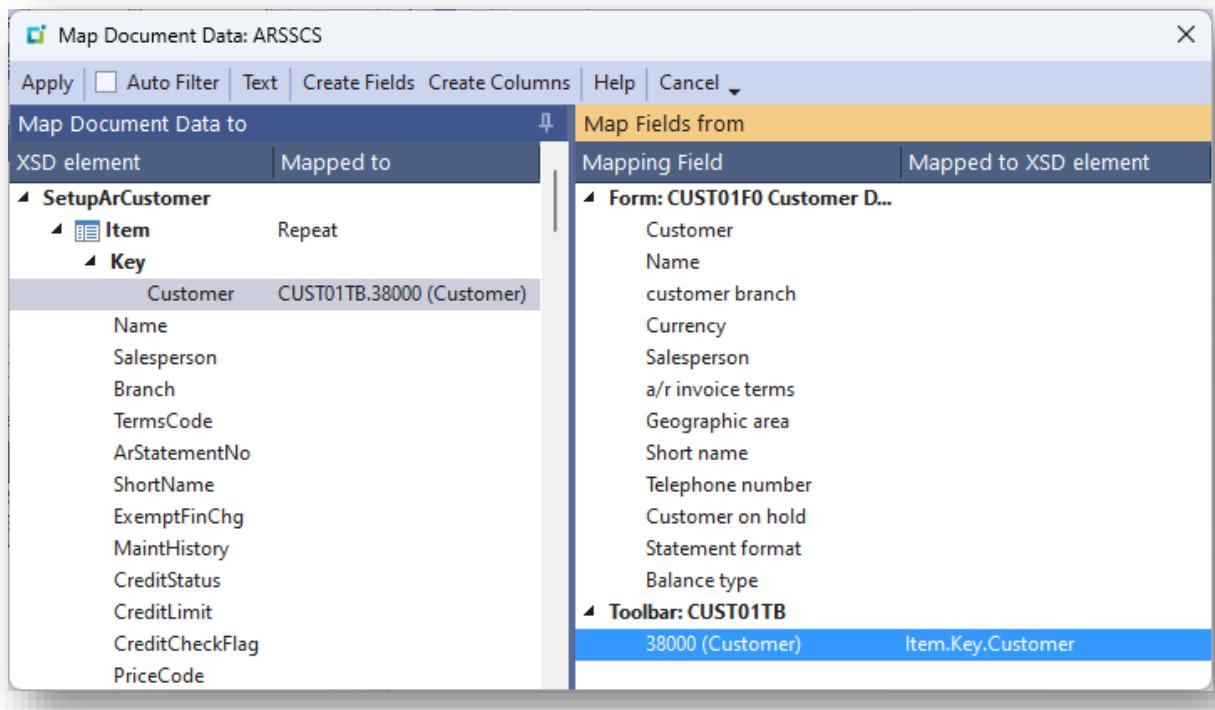
Note: You can also create form fields and listview columns from this mapping window, if you haven't yet pre-defined such fields. However, you have to have

created the form and listview controls in advance, even if they don't yet contain any fields or listview column headers.

When the Mapping window is shown any existing form fields, toolbar fields and listview column headers (in the application) will be indicated, like this:



Let's say that you wish the **Customer** field in the KEY XSD element to be derived from the value in the Customer toolbar control – just select the **Customer** element (the **Map Document Data to** window) and then double-click the toolbar **Customer** item in the **Mapping Fields** window:

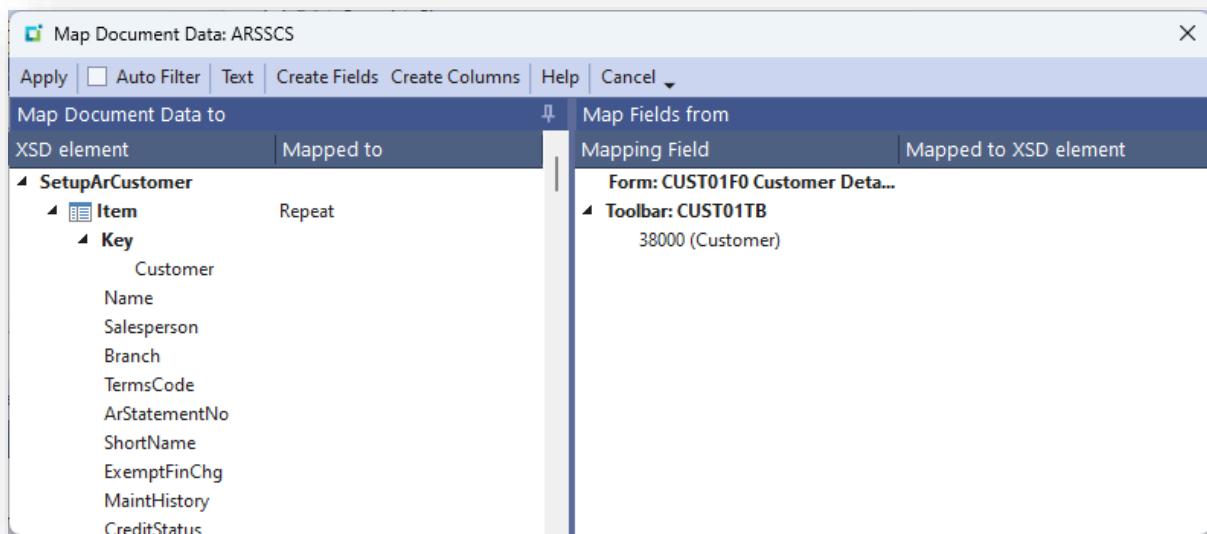


And that's it. When the transaction data sources is executed, the value in the toolbar Customer field is extracted and inserted into the XML document that will be created.

To remove the mapping just double-click the toolbar Customer item, or press **Del** on the XSD element.

5. You can create form fields and/or listview column headers automatically from any XSD element.

Let's say you haven't yet created any form fields BUT you have created an empty form called **Customer Details**. Your mapping window will look like this:



Now click on **Create Fields**. All unmapped XSD elements will be shown in this window:

Select	XSD element	Caption	Data type	Length	Decimals	Control to add to
<input type="checkbox"/>	Customer	Customer	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	Name	Name	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	Salesperson	Salesperson	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	Branch	Branch	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	TermsCode	Terms code	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	ArStatementNo	Ar statement no	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	ShortName	Short name	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	ExemptFinChg	Exempt fin chg	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	MaintHistory	Maint history	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	CreditStatus	Credit status	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	CreditLimit	Credit limit	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	CreditCheckFlag	Credit check flag	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	PriceCode	Price code	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	CustomerClass	Customer class	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	InvDiscCode	Inv disc code	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	BalanceType	Balance type	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	Area	Area	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	LineDiscCode	Line disc code	Alpha	0		Customer Details (CUST01FO)

Let's first add the **Name** element to the form. You will notice that for each field you can define the Caption, the Data type (Alpha, Numeric etc.) and the length, but you may not know the data type or indeed the length of the field that the business object requires. This is why a suggested list of items is shown:

Select	XSD element	Caption	Data type	Length	Decimals	Control to add to
<input type="checkbox"/>	Customer	Customer	Alpha	0		Customer Details (CUST01FO)
<input checked="" type="checkbox"/>	Name	Name	Alpha	0		Customer Details (CUST01FO)
Name Generic name (e.g. buyer) (Alpha 50)						
NameFull Full name (Alpha 100)						
NamePart Name component (first, middle, last etc) (Alpha 30)						
NameSuffix Name suffix (Alpha 20)						
NameTitle Name title (eg Mr Mrs Miss) (Alpha 10)						
NamedUserFlag (AdmOperator) No [N], Yes [Y] (Checkbox)						
<input type="checkbox"/>	CreditCheckFlag	Credit check flag	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	PriceCode	Price code	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	CustomerClass	Customer class	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	InvDiscCode	Inv disc code	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	BalanceType	Balance type	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	Area	Area	Alpha	0		Customer Details (CUST01FO)
<input type="checkbox"/>	LineDiscCode	Line disc code	Alpha	0		Customer Details (CUST01FO)

You can click on any suggested item that matches the business object's requirements. In this case, we'll select the **Name** suggestion:

<input checked="" type="checkbox"/>	Name	Generic name	Alpha	50 0	Customer Details (CUST01FO)
-------------------------------------	------	--------------	-------	------	-----------------------------

You may need to change the Caption, as required:

<input checked="" type="checkbox"/>	Name	Customer name	Alpha	50 0
-------------------------------------	------	---------------	-------	------

Next, select the **Balance type**:

<input checked="" type="checkbox"/>	BalanceType	Balance type	Alpha	0
-------------------------------------	-------------	--------------	-------	---

BalanceType (ArCustomer) Balance forward [B], Open item [I] (Dropdown list)
Balanced (EftPayFormat) No [N], Yes [Y] (Checkbox)

You will notice that the Balance type shown in the suggested list is of type **Dropdown list**.

<input checked="" type="checkbox"/>	BalanceType	Balance type	Dropdown	1 0	Customer Details (CUST01FO)
-------------------------------------	-------------	--------------	----------	-----	-----------------------------

Now click the **Create** button. This will add the two fields to your form, with the correct enumerated values:

Customer Details	
Customer name	
Balance type	<ul style="list-style-type: none">Balance forward[B]Open item[I]

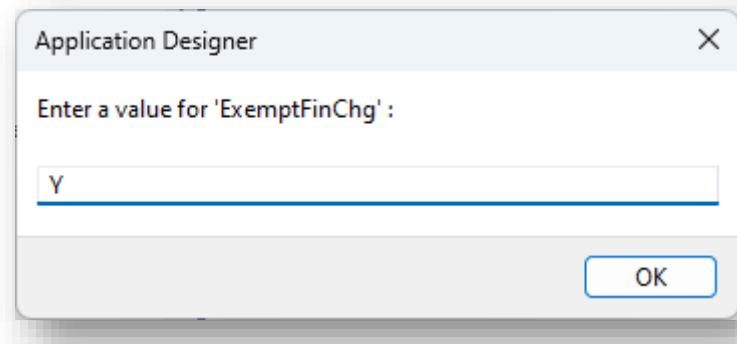
...and it will automatically map the two fields to the XSD elements:

The screenshot shows the 'Map Document Data' dialog box. On the left, under 'XSD element', there is a tree view of 'SetupArCustomer' which includes 'Item' (Repeat) and 'Key' (Customer, Name, Salesperson, Branch, TermsCode, ArStatementNo, ShortName, ExemptFinChg, MaintHistory, CreditStatus). On the right, under 'Mapping Field', there is a list of fields from 'Form: CUST01F0 Customer Details' (Customer name, Balance type) and 'Toolbar: CUST01TB' (38000 (Customer)). The 'Mapped to' column shows the corresponding XSD elements: Item.Name for Customer name and Item.BalanceType for Balance type.

Note: If you have already created more than one form, then you can select which form the field is to be added to:

The screenshot shows the 'Create Form Fields' dialog box. It lists four XSD elements: Customer, Salesperson, Branch, and TermsCode. The 'Control to add to' column shows dropdown menus where 'Customer Details (CUST01F0)' is selected for Customer and Salesperson, and 'Customer Details (CUST01F0)' is also selected for Branch. For TermsCode, 'Customer Details (CUST01F0)' is selected, while 'Credit Information (CUST01F1)' is also listed in the dropdown.

6. You do not need to map every single XSD element to a field as the business object will apply default values if an element is not supplied. However, you may wish to force a XSD element to have a specific value. To do this, first select the XSD element and then click **Text** (or right-click on the element) and this dialog will popup where you can enter the required text:



The value of the text will be shown in “ ” characters:

XSD element	Mapped to
SetupArCustomer	
Item	Repeat
Key	
Customer	
Name	CUST01F0.Customer name
Salesperson	
Branch	
TermsCode	
ArStatementNo	
ShortName	
ExemptFinChg	"Y"
MaintHistory	

7. While a data source can be executed automatically by linking it to an event, such as a toolbar field, there are times when you may need more control of the document or SQL statement that's about to be applied.

There are various script callouts (as indicated below) that can modify the document contents before a data source is executed:

Callout function	Syntax	Remarks
Execute a data source with SQL statement	(DataSourceExecuteSQL, "data source name", SQLStatement)	Executes the custom data source using the supplied SQL statement and returns the data.
Execute a data source with document	(DataSourceExecuteXMLDoc, "data source name", xmlDoc)	Executes the transactional Business Object data source using the supplied XML document string and returns the data. "0" will be returned if the data source did not execute successfully.
Get the XML document	(DataSourceGetXMLDoc, "data source name", " ")	Returns the content of the XML document for a transactional business object. While you can get at this document content in the OnBeforeTransaction event, you may need to get and view the content before executing the data source.
Validate a KEY value	(DataSourceValidateKey, "tableName or eventID", KEY + " xml node %key% not on file")	Returns a status indicating a KEY is valid. The KEY can be validated against a TableName (such as ArCustomer) or an EventID (such as 38000 indicating the Customer key). Return values can be DataSourceKeyValid, DataSourceKeyInvalid, "{1}" (invalid table name) or "{2}" (invalid eventID). Optionally you can supply the XML node to be returned from the TableName, in which case it will be the XML node value that is returned or a value of {3} if the XML node is invalid. Also optionally you can supply an error message that will be shown if the KEY is invalid. Note that compound keys can be supplied with commas as separators.
Change the type of transaction	(DataSourceSetupType, "data source name", "xxxxx")	Changes the type of transaction for a data source associated with a transaction 'Setup' Business Object. The type (Add, Update or Delete) can be selected from the popup list shown. Note that this changes the type for the duration of the application.

The script callout **DataSourceValidateKey** can be used to validate a KEY field, such as a Customer code or Stock code, is valid. You might need to use this callout, for example, to modify the Setup Type to be 'Setup Update' if a customer record exists, or 'Setup Add' if the record does not yet exist, by using the callout

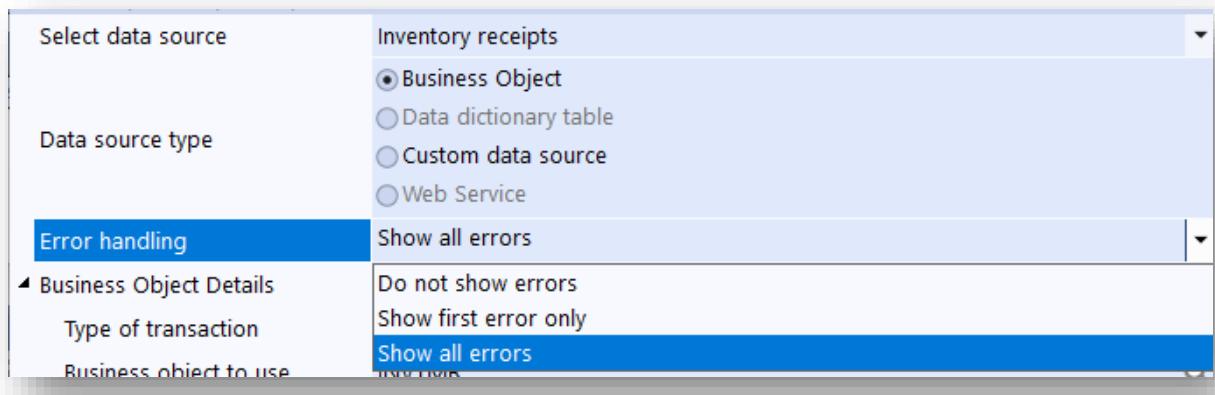
DataSourceSetupType.

And finally, there are two new script events that fire during the execution of a transactional data source – the **OnBeforeTransaction** event and the **OnAfterTransaction** event.

The **OnBeforeTransaction** fires just before the data source is about to be executed. This allows you access to the SQL statement (if it's a custom data source) or the XML document (if it's a business object data source) and you can then modify the document contents. You can also cancel the execution of the data source by setting the **Function** to be false.

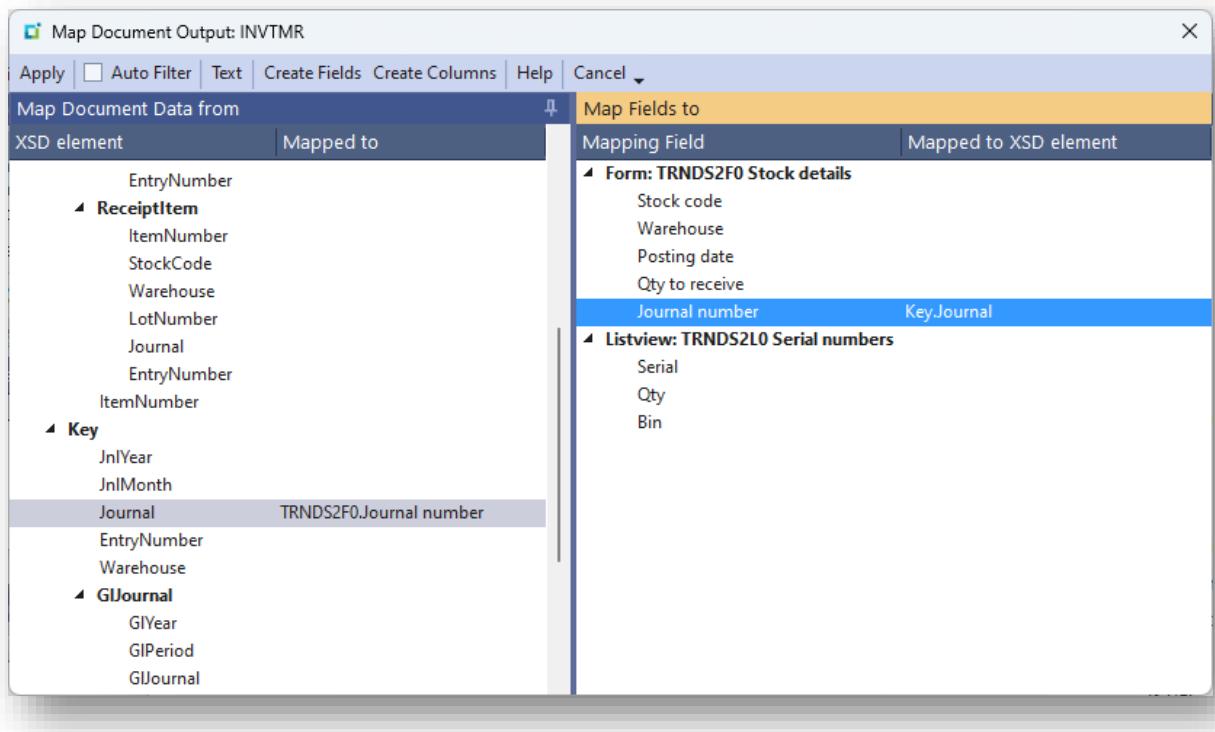
The **OnAfterTransaction** fires after the data source has executed and allows you access to any returned message being returned from a business object – such as error messages.

You can decide how the data source will handle transaction errors as shown in the **Error handling** options:



8. After mapping the document data, you can optionally map the document output from the business object.

As an example, using the business object INVTMR, the output contains the Inventory Journal number that was generated. You might want to show the journal number in a form field. In this case the mapping is the reverse of document data, in that you're mapping **from** the XML output **to** a field:



But alternatively, you can trap the **OnAfterTransaction** event and process the output from the business object and derive whatever values are required.

WEB SERVICE DATA SOURCES

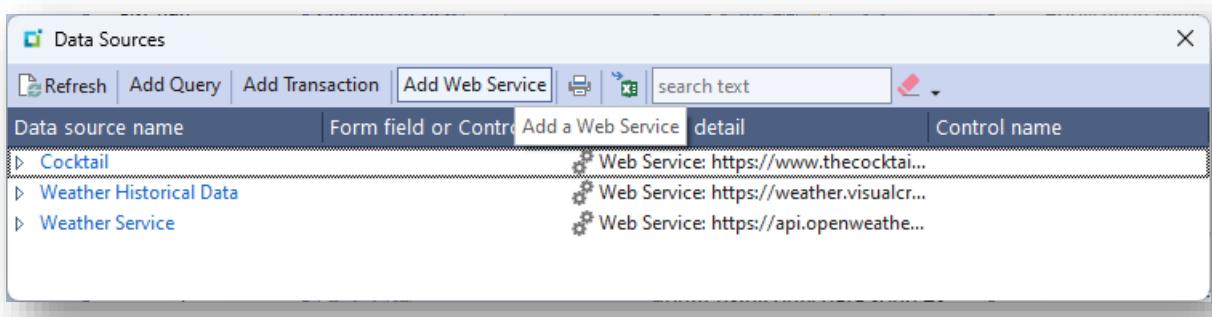
Web Service data sources allow you to easily communicate with web services using secure http requests. One of the main benefits is the ability to easily transform data from the web service to a format that's suitable for consumption by a SYSPRO application, such as being able to bind data to form fields and listviews. You may wish to look at the sample application **SAMSRV** to see how this can be achieved.

A web service can be executed automatically like any data source that is associated with an event, such as a toolbar field event or a form field changed event or when a listview is to be refreshed. Alternatively, you can execute the data source from script (see the callouts **DataSourceExecute** and **DataSourceRefresh**).

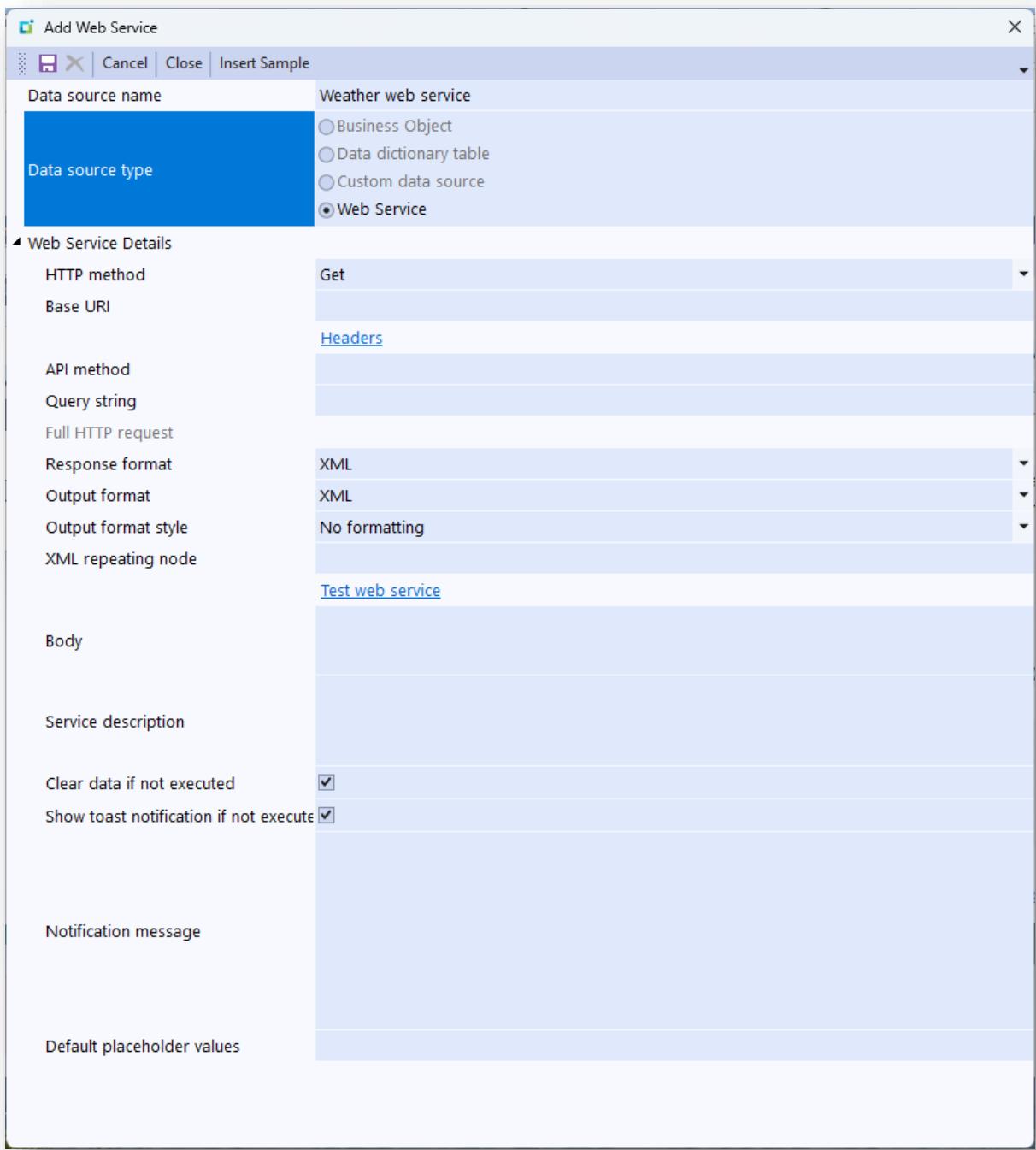
In addition, there are a number of script callouts that can alter the web service definition before a web service is executed:

Set the URL for a Web Service	(DataSourceWebSetURL, "data source name", URL)	Sets the URL for a web service data source. Note that this changes the URL for the duration of the applicaton.
Set the body content for a Web Service	(DataSourceWebSetBody, "data source name", BodyContent)	Sets the body content for a web service data source. Note that this changes the Body value for the duration of the applicaton.
Set a header for a Web Service	(DataSourceWebSetHttpHeader, "data source name", "header key value")	Sets a header key and value for a web service data source. If the header key does not exist it will be added to the list of headers. Note that this changes the Header for the duration of the applicaton.
Get the properties for a Web Service	(DataSourceWebGetProperties, "data source name", "")	Returns the properties of the data source as an XML string.
Get the URL for a Web Service	(DataSourceWebGetURL, "data source name", " ")	Returns just the URL of the web service.

To add a Web Service data source, click on **Add Web Service** in the Data Sources window:

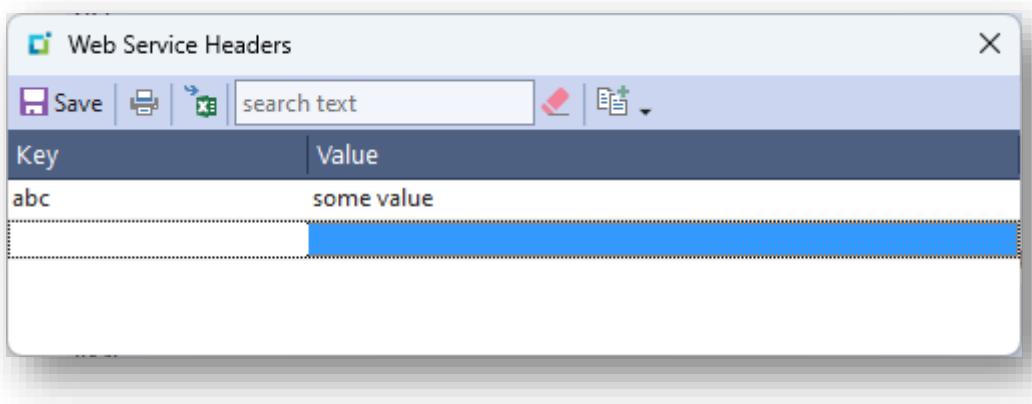


After entering a data source name, you will see these properties:



A web service is a http request which is made up of 3 parts – the base URI, the API method and the Query string. Together they make up the full HTTP request.

Some web services require additional information, and these are often supplied as **Headers** as a list of key-value pairs. Click on [Headers](#) to enter up to 20 Keys and Values:



The API method, Query string and Header strings can include special placeholders for values that will be injected into the strings when running the application. The placeholders include:

- %key% - indicates the primary KEY passed to the data source when executed.
For example, if the web service data source is executed as a result of entering a value into a toolbar field then the value of that toolbar field will be placed into the %key% placeholder.

Another 8 keys can be passed from the data source execution (%key1%, %key2%, etc.).

- [FormName.Caption] - the value of a named form field.
- [ToolbarName.ID] - the value of a toolbar field identified by its toolbar ID.
- \${GetVariable.VariableName} - the value of a variable created by script.
- \${GetGlobalVariable.VariableName} – the value of a global variable created by script.
- \${GetDatastoreVariable.Key} – the value of a datastore variable identified by its key.

Note that the web service will not execute if any of the variable placeholders do not exist.

Web services generally return their data either in XML or JSON format, which can be selected from the **Response format** option.

The **Output format style** determines how the data from the web service is to be handled by the application. You can select **No formatting** if you wish to handle the returned data yourself in the script, or you can select to bind the data directly to a form, a listview or a chart.

- **Form**
Select this if you wish the web service data to be bound directly to form fields.
- **Listview**
Select this if the data is to be bound to a listview. This would apply if the web service data contains repeating XML nodes. The data is converted to this format:
`<Query><Row>XML elements</Row></Query>`
The XML elements will be the children of the parent node specified in the **XML repeating node**.

- **Chart**

Select this if the data is to be bound to a chart.

- **No formatting**

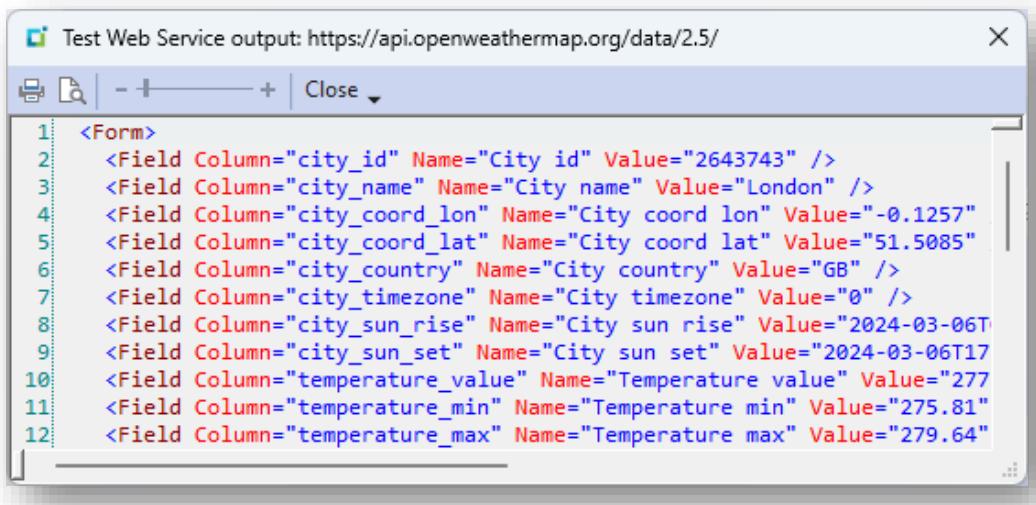
This returns the data as is without any formatting.

- **Flattened XML**

This returns the data as flattened XML document with a single root node and all elements and attributes as elements below the root node.

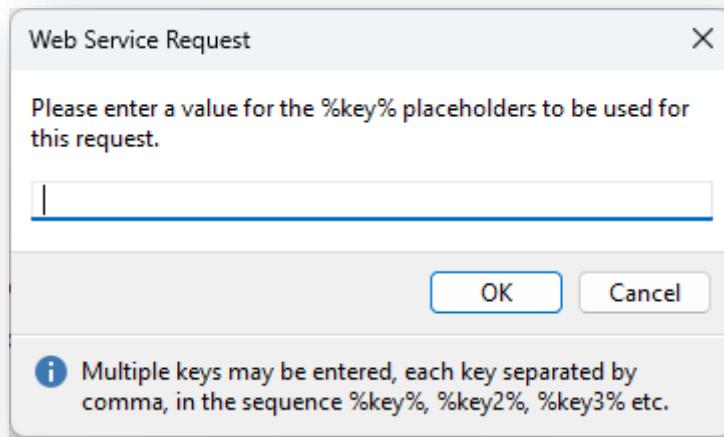
The property **XML repeating node** is important to define if the output format is Listview. Enter the full path for the repeating node. For example if the XML data contains a root node of **Current** and beneath that is a node of **Wind** and you would like to bind this element to a listview, then enter **/Current/Wind**. You can see this working if you click **Insert Sample** which will define a weather web service for you.

Click [Test web service](#) to try out the web the service. The returned XML or JSON will be shown in a window:



```
<?xml version="1.0"?>
<Form>
  <Field Column="city_id" Name="City id" Value="2643743" />
  <Field Column="city_name" Name="City name" Value="London" />
  <Field Column="city_coord_lon" Name="City coord lon" Value="-0.1257" />
  <Field Column="city_coord_lat" Name="City coord lat" Value="51.5085" />
  <Field Column="city_country" Name="City country" Value="GB" />
  <Field Column="city_timezone" Name="City timezone" Value="0" />
  <Field Column="city_sun_rise" Name="City sun rise" Value="2024-03-06T07:27:11" />
  <Field Column="city_sun_set" Name="City sun set" Value="2024-03-06T17:27:11" />
  <Field Column="temperature_value" Name="Temperature value" Value="277" />
  <Field Column="temperature_min" Name="Temperature min" Value="275.81" />
  <Field Column="temperature_max" Name="Temperature max" Value="279.64" />
```

Note that when testing the web service any placeholders described in the http request or headers will need to be entered, like this:



You can optionally enter default values for the keys in the **Default placeholder values** field so that you won't get prompted each time you test the web service.

If you have selected a HTTP method of POST, PUT or DELETE then you may wish to define the **Body** of the web service.

Optionally, you can describe the purpose of the web service in the **Service description** field. This is purely documentational.

Finally, you can decide how to handle errors returned from the web service.

Clear data if not executed

If checked and if the web service does not execute, then the values of form fields or listviews that are bound to the data source will be automatically cleared.

Show toast notification if not executed

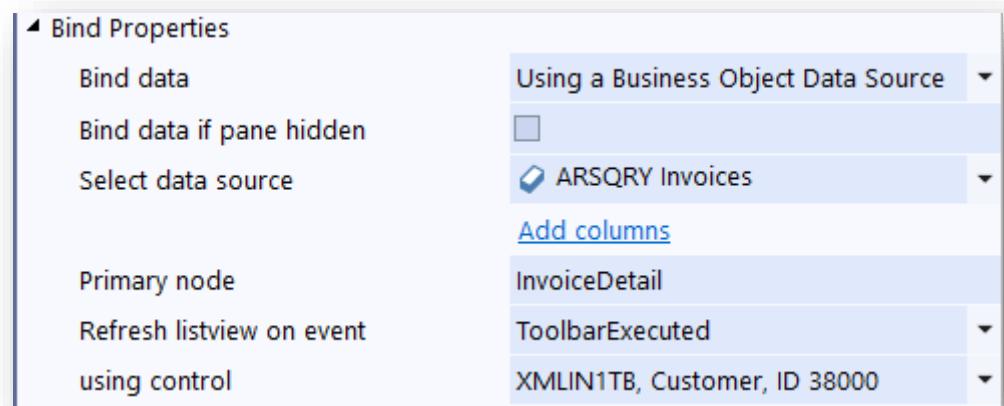
If checked and the web service does not execute, then a toast notification message is shown. The message content is defined in the **Notification message** property.

Notification message

This is the error message shown as a toast notification. Use the placeholder **%key%** to show the KEY used to execute the web service, and **%msg%** to show the error message returned from the web service.

USING A BUSINESS OBJECT DATA SOURCE FOR A LISTVIEW

You can bind data using a business object data source. This works in the same manner as using a Custom data source. You select a data source that must be associated with a business object, as shown below:

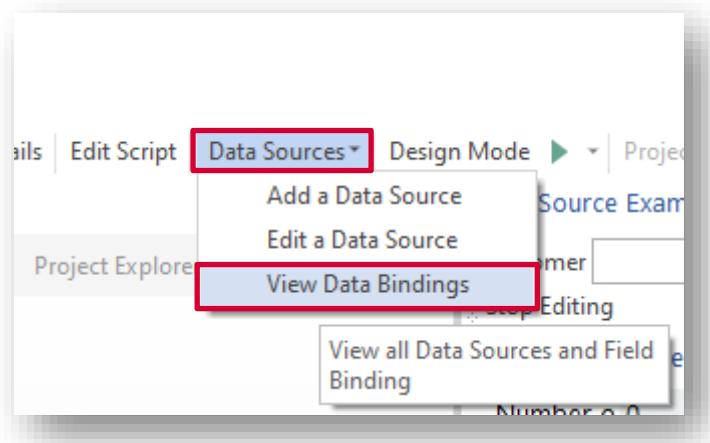


After selecting the data source the primary node will be automatically refreshed with the XML node defined against the data source.

This technique differs from the **Using a Business Object** option in that you will not have to define a script to be executed.

VIEWING DATA BINDINGS

On the main App Designer toolbar, you can view data bindings:



This will display all the form fields that are bound to data sources.

A screenshot of the 'Data Source Bindings' dialog box. The dialog has a header with 'refresh' and search icons, and a search text input field. The main area is a table with three columns: 'Data source name', 'Form field', and 'Data source detail'. The data is organized into sections with expandable arrows. The sections include 'AR branch' (Table Name: SalBranch), 'Customer' (Table Name: ArCustomer), 'Customer balances' (Table Name: ArCustomerBal), and 'Job Master' (Table Name: WipMaster). Under each section, there is a list of form fields, many of which are hyperlinked to their respective table names.

Click on the Data source name hyperlink to edit the data source.

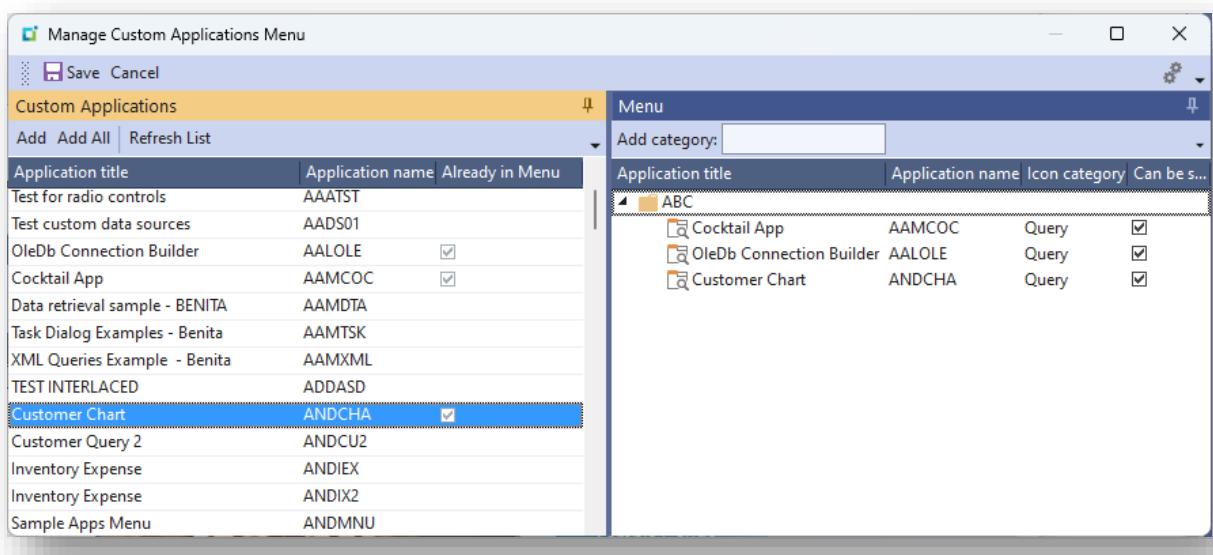
Click on the hyperlinked form field caption and this will navigate to the form field.

ADDING CUSTOM APPLICATIONS TO THE DESKTOP OR WEB UI MENU

Custom applications can be run as you would any typical SYSPRO program, but you may wish your application to appear on the **Program files** menu (in the Desktop UI) or the **Hamburger** menu (in the Web UI Avanti) just like a SYSPRO program.

You use the **Manage Custom Applications** program (**IMPID1**) to manage your menu.

This program shows all the Custom Applications found in the ...\\base\\AppDesigner folder on the server, and shows the current Menu structure (as defined in ...\\Plugin\\CustomerStore\\CUSMEN.IMP):



To add an item to the MENU list, select the item first then click on **Add** or double-click on the item. This will add the item to the top (first) category in the menu.

To remove an item from the MENU list, press the **DEL** key (note: you may find it useful to click on **Refresh List** as this will refresh the column 'Already in Menu').

To reposition an item in the menu, click and drag the item as required.

Add a new category as required (you can have as many categories as you like), then double-click an application item to attach it to that category.

When finished, click on **SAVE**.

Considerations:

- An application may have an ICON associated with it – select an appropriate one from the Icon Category dropdown list.
- By default, an application shown in the Menu structure can be secured by operator group or role. If an application can be freely run by anyone, then you should uncheck the **Can be secured** check box.

Applications that are marked as '**Can be secured**' will be shown in the **Group Maintenance** program where Access can be selectively allowed. Applications will be grouped under the module name **App Designer**.

For example, a menu structure like this, where the program AAMCOC is to be secured:

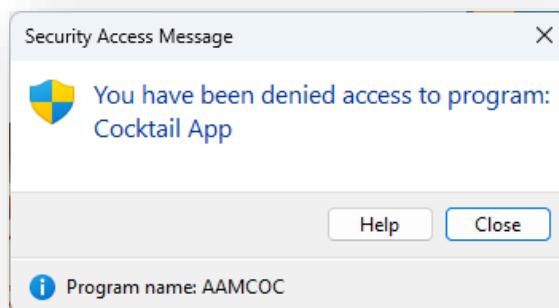
Application title	Application name	Icon category	Can be secured
Custom Applications			
OleDb Connection Builder	AALOLE	Query	<input type="checkbox"/>
Customer Chart	ANDCHA	Query	<input type="checkbox"/>
Cocktail App	AAMCOC	Query	<input checked="" type="checkbox"/>

will be reflected in **Group Maintenance** like this:

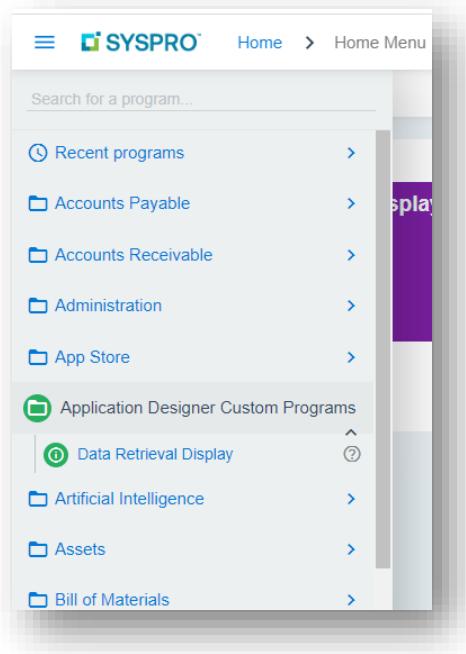
The screenshot shows the 'Operator Group Maintenance' window. A red box highlights the 'Group Details' section where the 'Code' is set to 'ADMIN' and the 'Description' is 'Administrator Group'. A red arrow points to the 'Security access' section, specifically to the row for 'AAMCOC'. In this row, the 'Access allowed' column has a checked checkbox, and the 'Module' column lists 'App Designer' and 'Cocktail App'.

Program	Access allowed	Module	Description	Browse only ...	Job logging re...
AAMCOC	<input checked="" type="checkbox"/>	App Designer	Cocktail App		

If access is disallowed for an application, the following message is displayed when a user tries to run it:



Tip: In Avanti, custom programs and modules on the Hamburger menu are indicated with green coloured icons:



VERSION HISTORY

You can document changes to the application using Version History.

Click on **Edit Application Details** and then click on the hyperlink **Version history**. This will show all previous documented versions:

Version History				
Version	Date	Modified by	Event	Comments
001	28/06/2023	John Smith	12345	Added script to control saving of the form.

Click on the **New** button to document the latest version:

Add Version

Cancel	
Version	002
Date	29/06/2023
Modified by	John Smith
Event	12346 Created a new data source to access the ArCustomer master table.
Comments	

Notes:

- You can edit or delete the last version added to version history, as long as the application is flagged as 'In development'.
- When you add or edit a version record then the **Release version** property is also updated in the main application form:

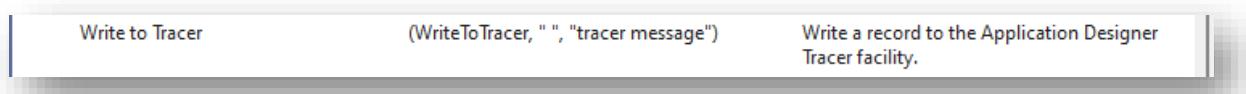
Properties	
Author	Fred Smith
Release version	003.1.1
Last modified	30/06/2023
Show notes in About window	<input type="checkbox"/>
Notes	
Category	

APPLICATION DESIGNER TRACER

The TRACER facility for Applications created using Application Designer is a diagnostic tool to help developers and support people monitor what an App Designer App is doing. When launched it will run in a SYSPRO instance for as long as that instance is running. The Tracer can be launched from within a running App or from the main SYSPRO menu system or within Avanti.

The Tracer can:

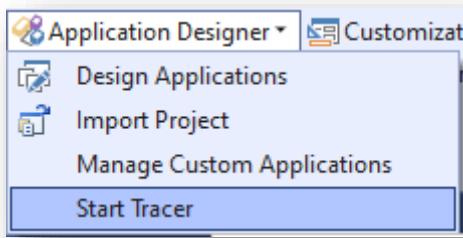
- Monitor the Application events showing the parameters passed to each event.
- Monitor every script Callout to the core SYSPRO product showing parameters being passed
- Display developer messages from within the VBScript using the WriteToTracer callout:



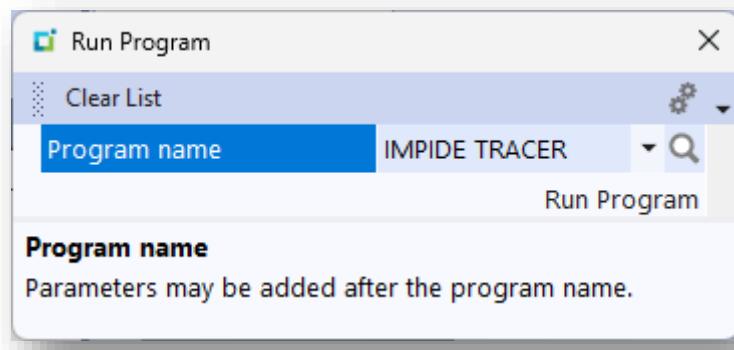
- Enable a developer to make temporary or permanent changes to the VBScript whilst the App is running.
- Track the use of script Callouts and warns if there appears to be excessive and recursive (incorrect) use.

The Tracer facility can be launched in the following ways from within SYSPRO:

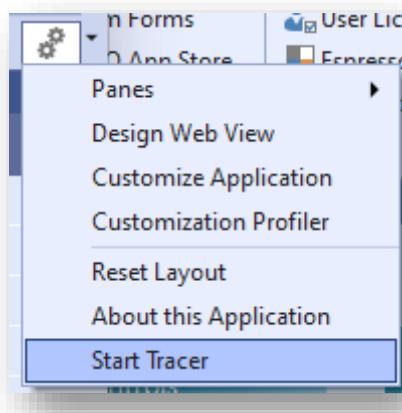
- From the Ribbon Bar:



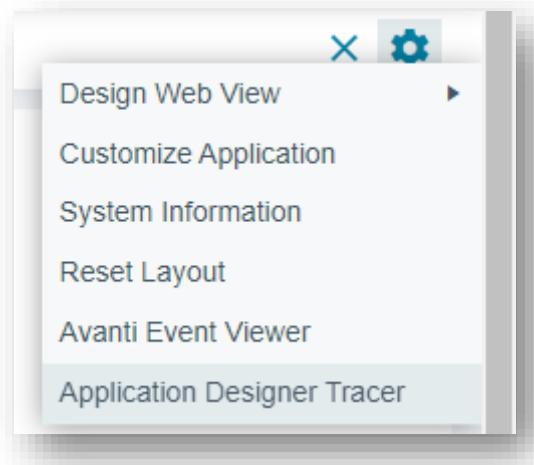
- By running IMPIDE with the parameter TRACER, as in this example:



- From the GEAR icon in programs designed with Application Designer:



In Avanti, the Tracer may be launched from the GEAR icon:



The Tracer window will be displayed like this in SYSPRO:

Application Designer Tracer									
Time	Control	Event	Value	Event ID	Caption	Callout	Data source	Result	Duration (secs)
9:22:35:30	DTS001TB	Toolbar	00000000000001	38000	customer		Customer	Success	0.23
9:22:35:67							Customer balances	Success	0.13
9:22:39:75							Use ARSQLY	Success	4.07
9:22:40:99							AR branch	Success	1.23
9:22:41:23							Customer balances	Success	0.23
9:22:41:25	DTS001	OnUserEvent							
9:22:41:30									

And like this in Avanti:

Application Designer Tracer - Running									
Start Pause Clear									
Time	Control	Event	Value	Even...	Capti...	Callout	Data source	Result	Dura...
10:44:48:21	DTS001TB	Toolbar	0000000000000001	38000	customer				
10:44:48:53							Customer	Success	0.30
10:44:48:71							Customer balances	Success	0.15
10:44:49:91							Use ARSQRY	Success	1.19
10:44:50:06							AR branch	Success	0.12
10:44:50:19							Customer balances	Success	0.12
10:44:50:20	DTS001	OnUserEvent							
10:44:50:22					Data s...				

To start the TRACER, click on the **Start** button. You may pause the tracer logging by clicking on the **Pause** button at any time – click **Start** to start the logging again.

USING THE TRACER IN SYSPRO

The Tracer in SYSPRO has some additional features over the Tracer in Avanti, as follows:

- The Tracer window can be ‘minimized’ by clicking on Minimize – the window will be reduced to just showing the toolbar:



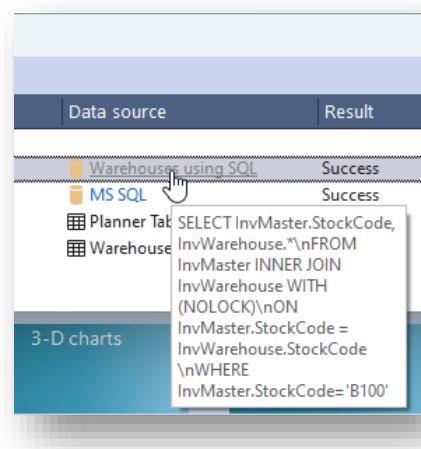
Click on **Restore** to restore the Tracer window to its previous height. When the Tracer window is minimized the tracer logging is automatically ‘stopped’.

You can also click on the X in the top right-hand corner of the window to minimize the window.

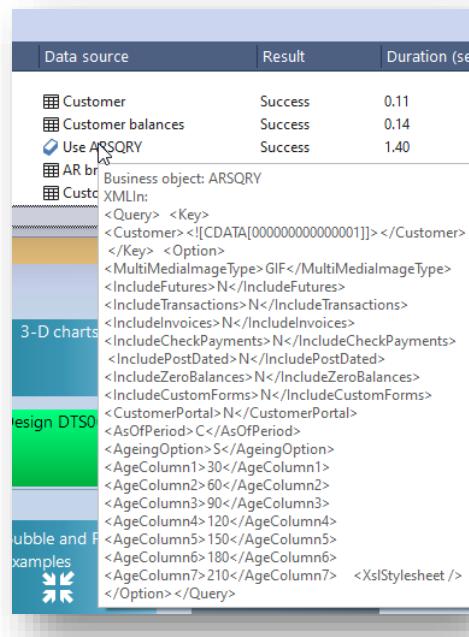
If you are in the main SYSPRO menu then the Tracer window will be hidden rather than minimized and the trace logging is automatically stopped (and the contents of the trace logging remains intact). You can start the Tracer again by clicking on the Tracer option in the Ribbon Bar.

- Move your mouse over a data source to see the statement that was executed. Click on the hyperlink to copy the tooltip text to the clipboard.

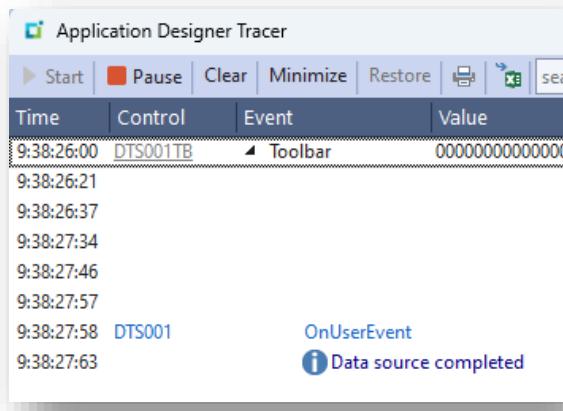
For a SQL statement you can then paste the clipboard contents directly into SQL Server Management Studio to test the statement.



For a business object data source the clipboard contents can be pasted into the Business Object Input of the e.Net Diagnostics tool:



- Click on the hyperlinked Control name to start editing the application in the Application Designer.



- Click on the hyperlinked Event message to edit the VBScript for the application:

```

VBScript for DTS001
Apply Insert Tracer | ⌂ ⌃ ⌄ ⌅

1 Option Explicit
2
3
4
5 Function AppDesigner_OnUserEvent(EventType, EventName, EventID, Param1, Param2, Param3)
6
7 if EventType = ToolbarEvent then
8   if EventID = 1001 then
9     call CallSYSPROFunction(FormStartEditing, "DTS001F3", " ")
10    call CallSYSPROFunction(FormFieldSetTooltip, "DTS001F1", "customer|Acme Brush Company")
11  end if
12  if EventID = 1002 then
13    call CallSYSPROFunction(FormStopEditing, "DTS001F3", " ")
14  end if
15  if EventID = 1004 then
16    returnValue = CallSYSPROFunction(DataSourceExecute, "Use ARSQRY", "1")
17    msgbox returnValue
18  end if
19 end if
20
21 if EventID = 38000 then
22  call CallSYSPROFunction(WriteToTracer, " ", "Data source completed")
23 end if
24
25 End Function
26
27
28

```

At any time you can click **Apply** – the changed script will be applied directly to the running application. In other words, you can see the effect of your changes as the program is running.

Additionally, if you are happy with the script changes you can click the **Save** button and this will immediately update your application .txt file with the changed script.

Note that this feature is only available under the following conditions:

- The script must not have any syntax errors
- You are allowed access to the program IMPIDE
- The application that is running is not encrypted
- The application.txt file is not read-only
- You are not running in another instance of SYSPRO

GETTING HELP

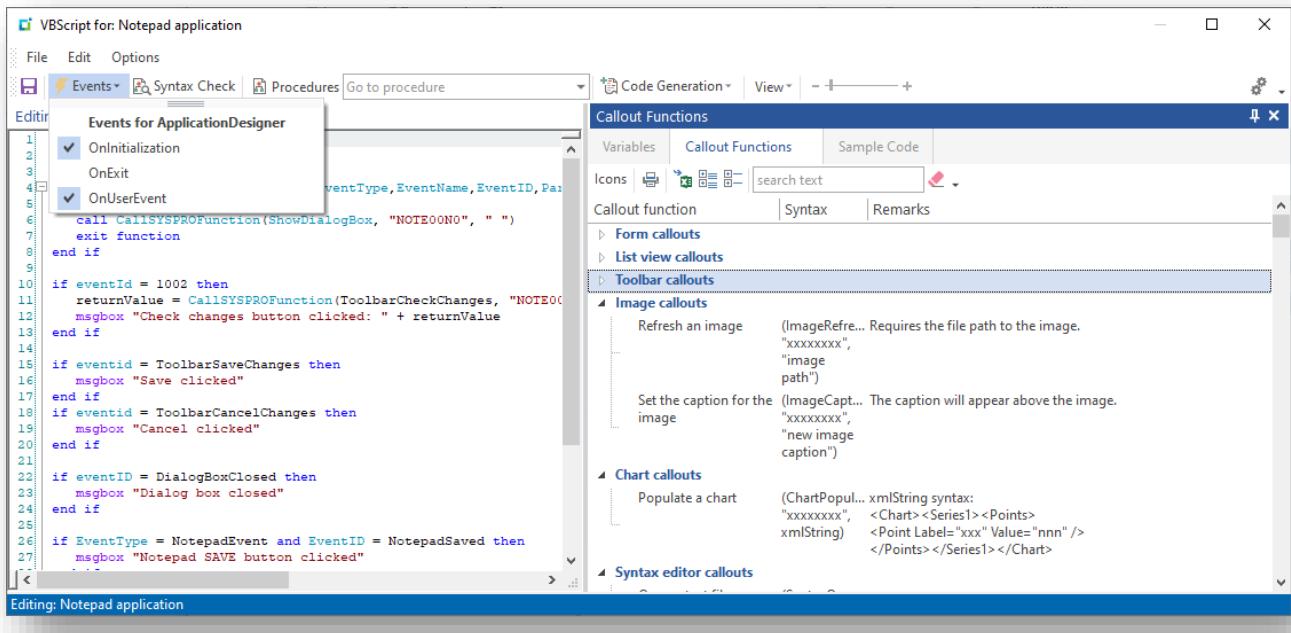
This guide, the Help panel built into the Designer, plus the **Application Designer** movies will get you up to speed with designing and deploying applications.

You may also consider visiting our Forums page at <https://forums.syspro.com/>.

Programming the Application

The application's behavior and logic can be controlled by script. The script contents are embedded in the application when the application is saved.

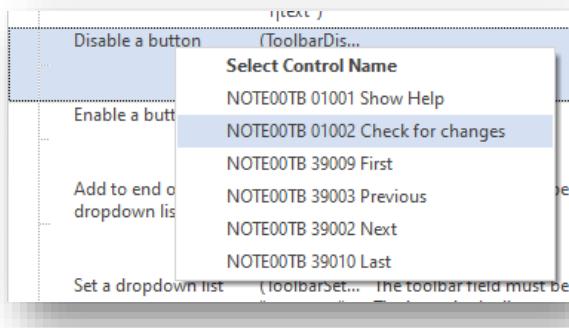
To edit the script, click on the **Edit Script** button in the main toolbar. The script editor window will look something like this:



The editor consists of a main text control to enter the script and a variety of other panes that help create script content. The main panes are covered here:

1. Callout functions

This pane contains all the callouts that can be invoked by a script. These callouts affect the control and logic of the entire application, from disabling toolbar fields to populating a data grid. To apply any callout, simply double-click on it and a context menu will popup showing all the controls that are appropriate for that callout:



Click on the required control name and the callout statement will be injected into the script editor at the current text position.

2. Variables

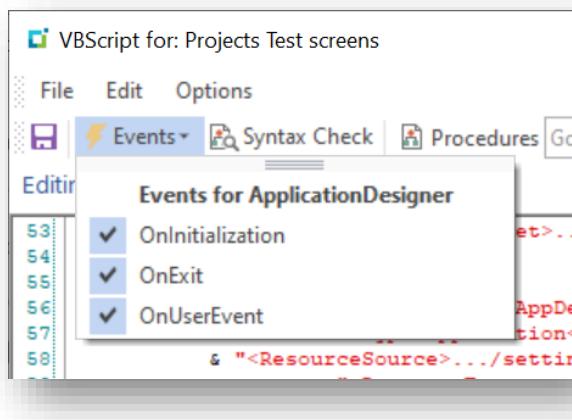
These contain various read-only (and occasionally read-write) variables that you can use in your code, anything from the current company code to various folder paths.

3. Sample code

This pane contains example code that can be inserted into the script.

APPLICATION DESIGNER VBSCRIPT EVENTS

An application can raise one of three events each time the UI is instructed to do so:



OnInitialization and **OnExit** events do not take any parameters and are raised when the application is initialized and exits respectively.

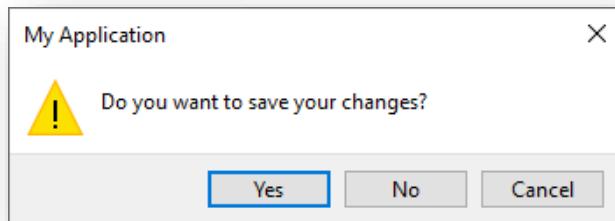
SCRIPTING THE **ONEXIT** EVENT

The **OnExit** event can be used to prevent a user from inadvertently exiting out the application by setting the **AppDesigner_OnExit** function to be **false**.

Here's an example of how to use this:

```
AppDesigner_OnExit=False
returnValue = CallSYSPROFunction(TaskDialog, " ", "<Msg><Title>My
Application</Title><Text>Do you want to save your changes?</Text><Footer Icon=''
Text='' /><Options /><Buttons Default='002'></Buttons><Icon>warn</Icon><Standard
Cancel='true' Yes='true' No='true' /></Msg>")
dim TaskDialogArray
TaskDialogArray = split(returnValue,vbTab)
if TaskDialogArray(0) = 2 then
    AppDesigner_OnExit=true
end if
End Function
```

When this application runs and the user tries to close the application, the following message will appear:



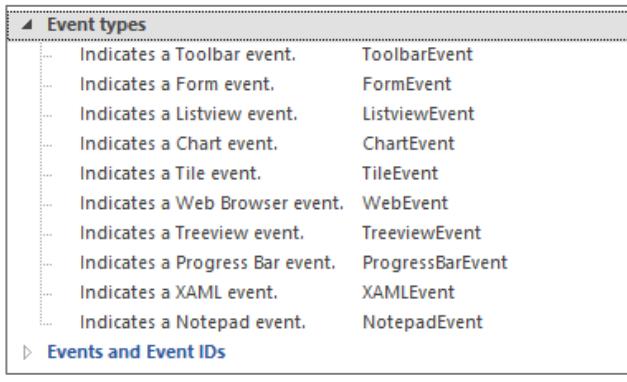
If the user selects **Yes**, then the application will exit, otherwise it won't.

SCRIPTING THE **ONUSEREVENT** EVENT

The most used event is the **OnUserEvent**, which is raised any time a user interacts with the UI. This event has the following format:

```
Function AppDesigner_OnUserEvent(EventType, EventName, EventID, Param1, Param2, Param3)
```

These parameters are explained as follows:

Parameter	Explanation
EventType	This indicates the type of control that invoked the event. These event types can be seen in the Callout pane of the VBScript editor: 
EventName	This is usually the name of the pane that the control is on.
EventID	This is the unique id of either the toolbar button executed or otherwise the specific type of event. You can view all the events that can fire in the Events and Event IDs section in the Callout pane – see sample image below.
Param1,2,3	These hold the data relevant to the type of event.

Events and Event IDs	
Fires when a form field value changes.	FormValueChanged
	Param1 contains the form field caption (in lower-case), Param2 contains the changed value and Param3 contains the type of field changed (A=Alpha, B=Check box, D=Date, E=Color, F=Font, I=Picture, 1=Multiline, 2=Spin, 3=Slider, 4=RadioButton, 5=Hyperlink, N=Numeric, L=DropDown, K=DropDownEntry). Example: If EventType = FormEvent and EventID = FormValueChanged then Param1 contains the field caption. Note that the OnGainFocus event must be checked against the Application Details for this event to fire.
Fires when a form field has focus.	FormGainFocus
Fires when ENTER pressed.	FormENTERPressed
Fires when a listview row is selected.	ListRowSelected
Fires when right-click on a listview row.	ListRowRightClick
Fires when double-click on a listview row.	ListRowDoubleClick
Fires when a listview cell value changes.	ListCellChanged
Fires when clicking on a listview hyperlink.	ListHyperlinkClicked
Fires when a listview check box is changed.	ListCheckBoxChanged
Fires when the DEL key has been pressed on a listview row.	ListDELPressed
Fires when a toolbar button has focus.	ToolbarGainFocus
... Fires when clicking on a chart point.	ChartClicked
ent")	Param1 contains the chart series, Param2 contains the chart point.

To add an event into the VBScript, select **Events** from the toolbar menu, followed by the event that you need. It is logical (and sensible) to put the events in sequence in your script (e.g. **OnInitialization**, **OnUserEvent**, **OnExit**).

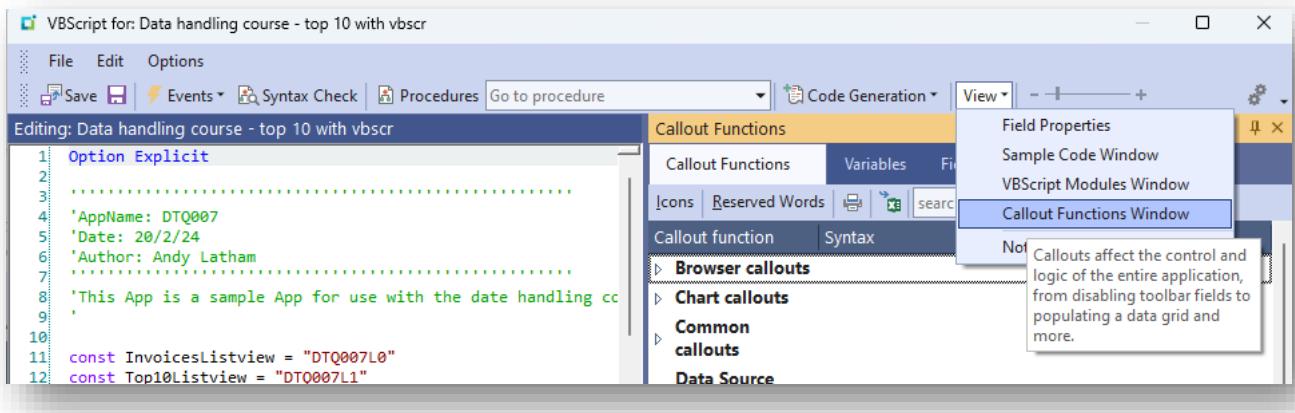
Note: VBScript events only execute when the application is running. They do not fire when you are designing an application.

APPLYING A SYSPRO CALLOUT IN SCRIPT

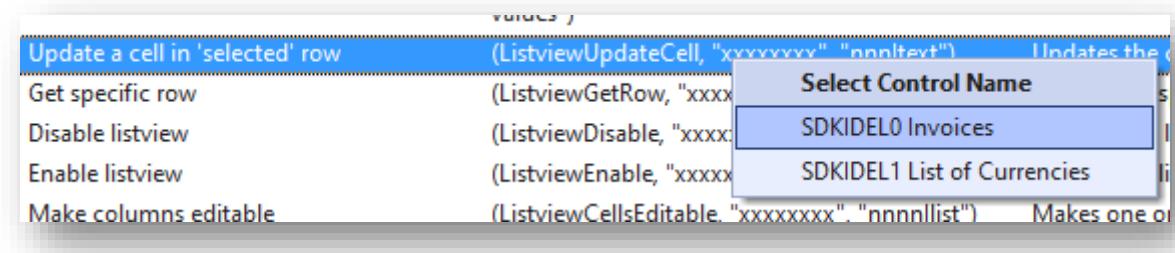
An application's business logic can be partially controlled using data sources and other available techniques, but sometimes the complexity of the application requires some programming logic to be applied.

This is when using a script becomes important to control the behavior of the application. In particular, being able to change the content of a control (such as Listview or Form) or to interrogate the contents of a form field or listview row is essential. And this capability is handled by the SYSPRO callouts.

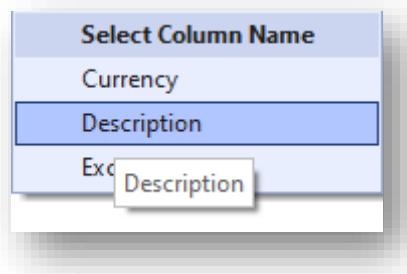
There are many such callouts available to the script – click on View > Callout Functions Window to see a complete list:



To make use of a callout and insert it into your script, just double-click on the callout. You will be prompted to select the control name that's available in your application. For example, the Listview callouts all require the name of the listview, so when you double-click you will be presented with the titles of the listviews in your application – simply select the control name and the callout function will be inserted into your script.



Some controls require further selection to help you construct the appropriate callout string. For example, some listview callouts require you to supply the Column name or index and in this case the selection of available columns will be presented:



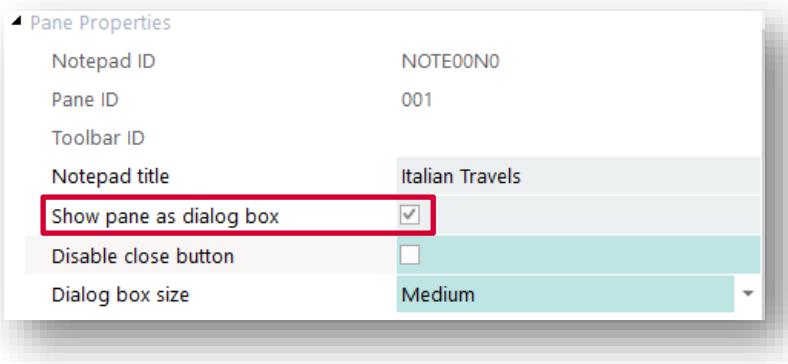
After selecting a column name the callout will be inserted into the script with the correct column index and a comment reflecting the column name:

```
call CallSYSPROFunction(ListviewUpdateCell, "SDKIDEL1", "002|text") ' Description
```

HOW TO USE A DIALOG BOX

Dialog boxes in an application can be a useful way to present information in a popup window that the user must then close to continue processing.

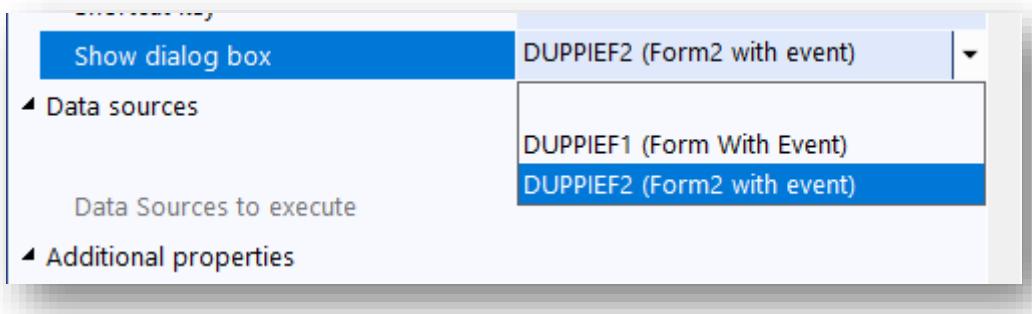
In the Application Designer, any control and its associated toolbar can be presented in a dialog box. To make a control appear as a dialog box, enable the **Show pane as dialog box** option:



This option ensures that this pane will be automatically hidden (and disabled) as the application loads, regardless of how the pane appears while designing the application.

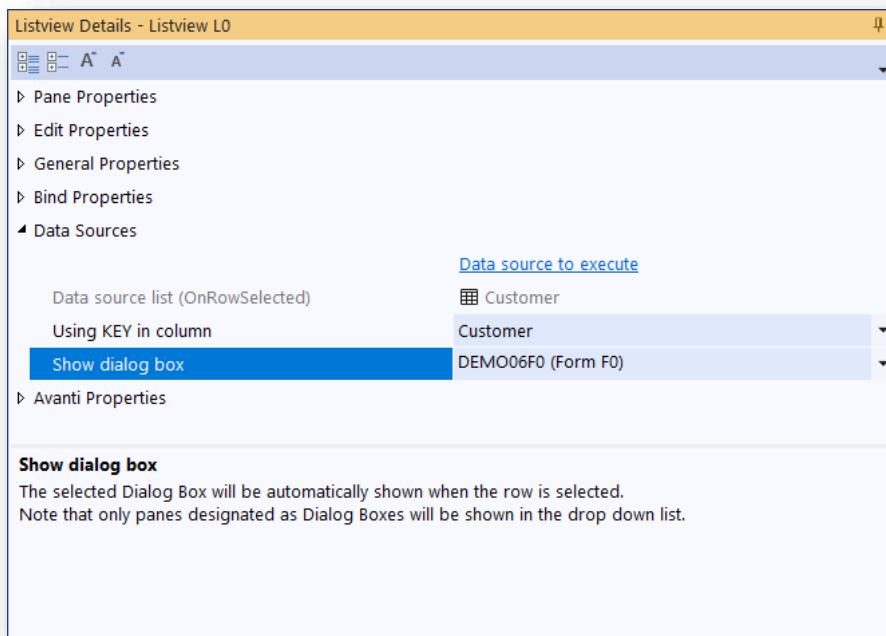
One way to show the dialog box is via code, using the script callout **ShowDialogBox**. When this callout is invoked, the pane is shown as a modal window and interaction with the main application is prevented until the dialog box is closed.

Alternatively, you can make a toolbar button automatically show a dialog box using the property **Show dialog box** when designing a toolbar.



Note: Only panes that are designated as Dialog Boxes will be shown in the drop-down list. At runtime, when the button is clicked the dialog box is first shown before any other actions are taken (such as executing a data source and the OnUserEvent in the script).

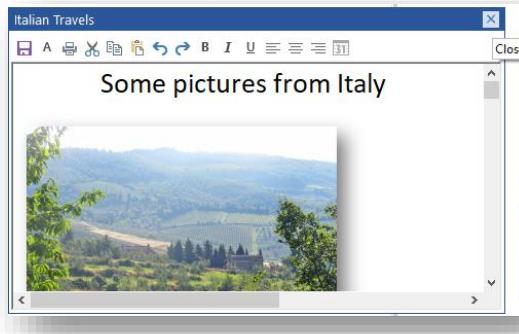
You can also show a dialog box automatically when a row is selected in a listview:



HOW TO CLOSE A DIALOG BOX

There are two ways a dialog box can be closed:

1. By the user clicking on the X in the top right-hand corner of the window:



2. By using the script callout **CloseDialogBox**.

You may wish to prevent the user from exiting out of the application by clicking the X, so you can select the option **Disable close button**. This means that the only way a dialog box can be closed is via code.

Typically, you would have a CLOSE or SAVE button in the control's toolbar and have code behind these buttons that then invokes the **CloseDialogBox** function.

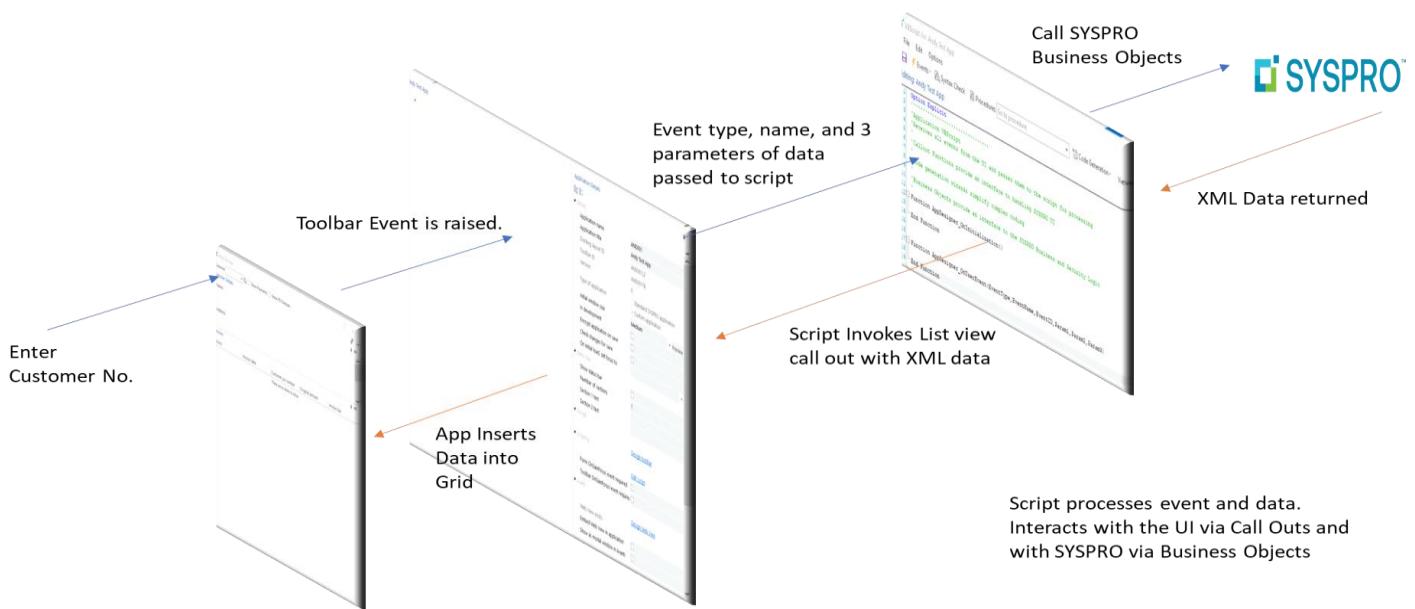
Note: When a dialog box is closed the event **DialogBoxClosed** fires. You might use this event to set focus to a form or toolbar.

You can also define the initial size of the dialog box with the option **Dialog box size**. This is just the initial size to be presented to the user; thereafter the system will remember the position and size of the dialog box.

EVENT CYCLE PROCESSING

This diagram shows how events are typically processed in an application.

1. The user enters a Customer Number in a toolbar.
2. A toolbar event is raised which is intercepted by the application's script; the script calls a business object which in turn returns XML data.
3. The script processes the XML data and issues a callout to populate a listview with the XML data.
4. The listview is populated with data, the script exits, and control is passed back to the application.



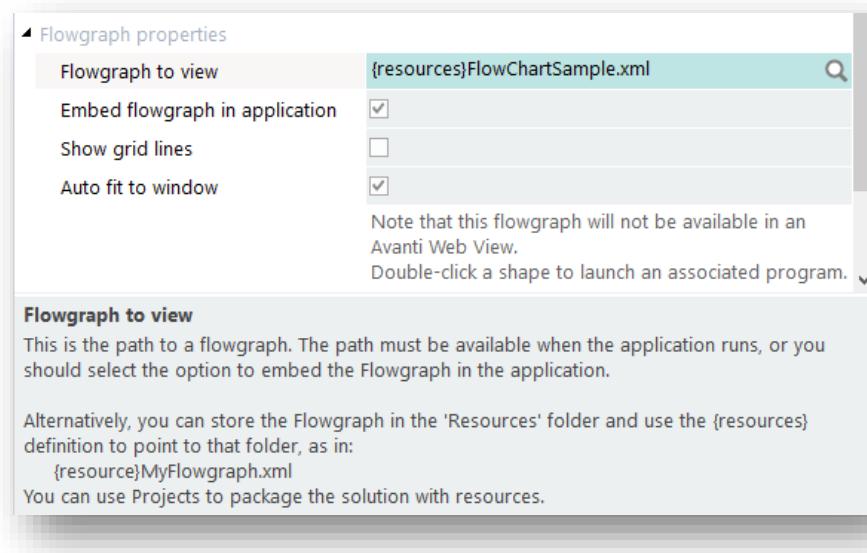
SOME BASICS

Important information regarding application file names and folder locations:

- Applications have a 6-character application name.
- Applications reside as text files in the SYSPRO .../base/settings/AppDesigner folder.
- Project files reside in the .../base/settings/AppDesigner/Projects folder.
- There is a .../base/settings/AppDesigner/Resources folder which can be used to hold any special resources that the application may need.
- In a client/server installation, the Applications in the AppDesigner folder and the Resources folder are synchronized with the main server.

Note: The Projects folder is not synchronized: this is a personal development tool for a developer and as such, resides on each developer's local machine.

- Avoid hard coded folder names and always use relative folder names. In this way, the application does not become bound to a specific SYSPRO installation.
- The VBScript editor exposes the following useful System Variables that help construct relative folder names:
 - BaseSettingsFolder
 - Company
 - MultiMediaPath
 - VBSModulesPath
- The Application Designer also exposes a special keyword syntax for pointing at the Resources folder using the keyword **{resources}** in the path for various controls:
 - Flowgraph
 - Notepad
 - Image viewer
 - Document viewer
 - Syntax Editor
 - XAML control



- The **{resources}** keyword can also be used in VBScript when setting a control's filename.

```

Function AppDesigner_OnInitialization()
    dim filename
    filename = "{resources}a100.pdf"
    call CallSYSPROFunction(DocumentShow, "NOTE00PO", filename)
End Function

```

Note: The interpretation of **{resources}** is done by the control not by the VBScript.

- The operating system temporary directory can be used to contain temporary files (see later).
- An application named **SAMXML** will generate a file called `SAMXML_AppDesigner.txt`. It is important therefore that a naming convention is established so that if applications are being written by multiple developers, there is no clash in application names. This is particularly important when using the project export and import function. It would be very easy to overwrite an application written by someone else.

This goes for any component such as .Net User Controls, and VBS modules.

Some VBScript Suggestions

This section tries to provide pointers to what we have found to be good practice.

Because the Application Designer is so easy to use and writing VBScript is not onerous, thought should be given to other developers who may either need to enhance the code or need to debug it should there be a problem.

FUNCTIONS AND CLASSES

- Write code in re-useable functions and pass parameters to them.
- Do not write monolithic pieces of code. They will be difficult to debug later.

COMMENT THE CODE

- Introduce the code with a description of who wrote it, when and what it is supposed to do.

For example:

```
1 Option Explicit
2 ****
3 'Name: SAMDTA
4 'Date: 8/3/2022
5 'Authored by: Andy Latham on behalf of SYSPRO Ltd
6 ****
7 'This is a working example app that demonstrates three ways to retrieve a list
8 'of customer invoices for a specific customer from Syspro and populate a list view
9 '1. By using a Syspro Business Object
10 '2. Using a direct SQL query using ADODB to return XML
11 '3. Using a direct SQL query using ADODB to return an ADODB recordset, building XML in the script
12 '
13 'Please note that the recommended mechanism to get Syspro data is to use the Syspro Business Objects
14 'The business objects execute all the in-built security and business logic that Syspro uses
15 'Direct SQL calls are not recommended to access Syspro data because it bypasses the business logic
16 'but a direct SQL call may be required to access data in other Non-Syspro databases
17 '
18
```

- Make notes in the code to explain what functions do.

For example:

```
334 ****Function that queries Database, returns a recordset and builds XML to send to the listview
335 function UpdateListView_ADODB_Recordset(CustomerNumber, servername, dbname, writexmltofile, showdebugmessages)
336 'first setup the class to get access to the database
337 'to use the FOR XML AUTO query to get XML returned from the SQL database
338 Dim db
```

APPDESIGNER_ONUSEREVENT

This event is the most critical of events, as every control will raise this event when actions are performed on them.

It is useful to be able to see what data is being passed into the event by a control. Therefore, the following code is an example of how to do that:

```
dim msg
msg = ""
msg = msg + "EventType: " + EventType + VBCRLF
msg = msg + "EventName: " + EventName + VBCRLF
msg = msg + "EventID: " + EventID + VBCRLF
msg = msg + "Param1: " + Param1 + VBCRLF
msg = msg + "Param2: " + Param2 + VBCRLF
msg = msg + "Param3: " + Param3 + VBCRLF
msgbox msg
```

See [Appendix](#) for code snippet.

THE USE OF CONSTANTS

The Application Designer allocates unique EventIDs to Toolbar controls, which are then passed back the VBScript. Using these EventIDs, the VBScript can identify what the control was and make appropriate actions.

See above for the **OnUserEvent** parameters.

It is important to structure the code in such a way as to make it easy to read. Therefore, allocating constants with sensible names that refer to the toolbar button EventIDs helps readability and maintenance.

For example:

```
22
23  'toolbar event ids to help make code more readable
24  const TBar_Help      = "01001"
25  const TBar_DataRetrievalMethod = "01002"
26  const Tbar_CustomerNumber = "38000"
27  const Tbar_ShowDebugMessages = "01015"
28  const Tbar_WriteXMLtoFile = "01016"
```

Catching the **EventType** and then using a select statement to catch which control was pressed works well:

```

90 'Catch toolbar events
91 if EventType = ToolbarEvent then
92     select case EventID
93
94     *****Customer selected in combobox
95     case Tbar_CustomerNumber
96         CustomerNumber = param1
97         call RetrieveData(CustomerNumber, dataretrievalmethod, sqlserver,sqlDB)
98
99     *****show Debug Messages checkbox clicked
100    case Tbar_showDebugMessages
101        chkstatus = param1
102        if chkstatus = 1 then
103            showDebugMessages = true
104        else
105            showDebugMessages = false
106        end if

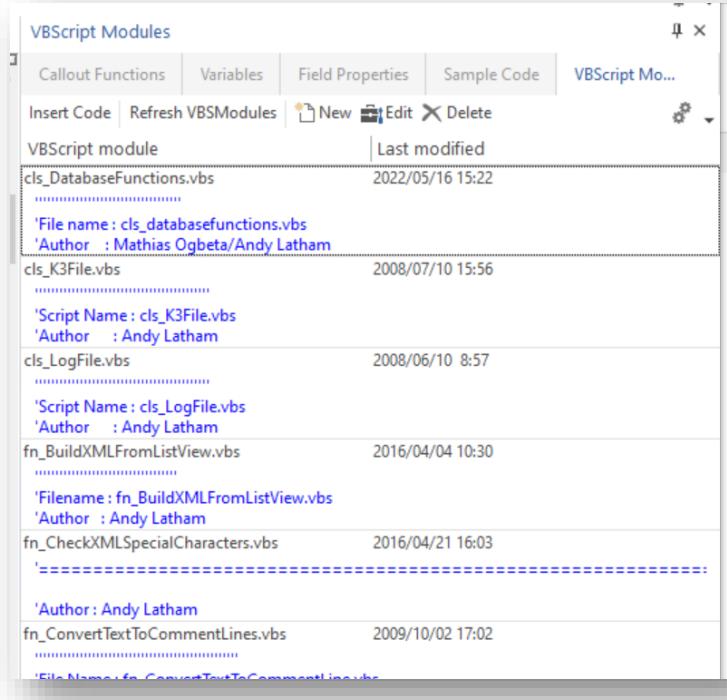
```

See [Appendix](#) for code snippet.

RE-USEABLE VBSCRIPT FUNCTIONS AND CLASSES

When developing several applications, there will inevitably be script functions and classes that are used across applications.

Try to avoid cutting and pasting common code from one application to another and use the VBS modules functionality within SYSPRO. The **LoadModule** function will inject the code into the VBScript before it is executed.



If you double-click on a VBS module, then this code is inserted.

At run time, the code in the module will be inserted exactly where the **LoadModule** call is made.

```
LoadModule ("cls_LogFile.vbs")
```

DEBUGGING AND LOGGING PROGRESS

The Application Designer does not have a VBScript debugger that will allow stepping through of code to check logic and contents of variables.

The best way to do this is using either a msgbox command to display data, or the Application Designer Task Dialogue box (see Sample App SAMTSK for examples).

As well as displaying progress, it may be useful to write out a data string (e.g. XML or XAML) being built so that it can be copied into other programs for testing syntax.

For example:

Generating XAML for a XAML pane, output the constructed XAML and use Visual Studio Blend to see what it looks like.

Please read the documentation on the XAML control for more information.

In both cases, it is often worth putting a flag in the code to be able to turn logging on and off. If it is in place to help develop the App it may be required later to debug it and often by other people so, make it easy for them.

Remember, if writing out files, make sure that the folder name is either relative to where the application is run or use SYSPRO variables to construct it.

For example:

```
filename = SystemVariables.CodeObject.baseSettingsFolder & "SAMDTA_ADOBE_XML.XML"
```

Or

```
35| Set fso = CreateObject("Scripting.FileSystemObject")
36| tempdir = fso.getspecialfolder(2)    'gets local temp directory
```

See [Appendix](#) for code snippet.

Also check VBS modules for a class called `cls_LogFile.vbs`.

USING PERSISTED VARIABLES

Remember that the VBScript events are raised by the UI each time a control is interacted with. This means that the VBScript is run each time an event occurs.

It is therefore not possible to set a variable, give it a value when one event occurs and then read it when another event occurs. All variables will be initialized each time the script is run.

There are two ways of setting variables that persist for the duration of an application:

1. Using system global variables:

There are 4 Global variables in the System variables pane that can contain string data and that are accessible during the run of SYSPRO.

You should use this carefully as such variables can be used by other applications and may overwrite your variables.

2. The preferred method is to use the VBScript callout functions **SetVariable** and **GetVariable**.

During the execution of a script, you may want to save a string as a variable and have that variable available during another script execution during the same run of the application. This is achieved using the callout **SetVariable**.

The callout allows you to name a variable (up to 20 characters long) and to provide a string of indeterminate length.

Set a variable	(SetVariable, "name", string)	Sets a named variable string.
Get a variable	(GetVariable, "name", " ")	Returns a named variable string.

You can have as many variables as you like. When the application has exited, the variables used for that application during the run of the application are removed.

You can at any time retrieve the variable string using the variable name with the callout **GetVariable**, in which you just supply the variable name, and the returned value is the variable string.

You can also set and get named **global** variables using callout functions in VBScript. Global variables persist for the duration of SYSPRO and are only removed when exiting SYSPRO or changing companies.

Set a global variable	(SetGlobalVariable, "variable name", string)	Sets a named global variable string. These variables persist for the duration of SYSPRO. A variable name cannot exceed 20 characters long. The string may be of any length. It is recommended to prefix your variable name with your application name or part thereof to uniquely identify it.
Get a global variable	(GetGlobalVariable, "variable name", " ")	Returns a named global variable string.

PASSING PARAMETERS TO AN APPLICATION

An application may be passed a parameter as part of an external command line or when running the application within SYSPRO. The parameter is retrieved by the application using the VBScript callout **GetAppParameter** – you will find this callout in the **Common Callouts** section:

Get application parameter	(GetAppParameter, " ", " ")	Returns the value that was passed as a parameter to the application.
---------------------------	-----------------------------	--

Typically, you would retrieve this parameter in the **OnInitialization** event, but it can be retrieved at any time during the run of the application:

```
Function AppDesigner_OnInitialization()
returnValue = CallsYSProFunction(GetAppParameter, " ", " ")
msgbox returnValue
End Function
```

To create the parameter from a command line, use this example syntax (assuming you are running SYSPRO from the ...\\base folder):

SYSPRO.exe /h /oper=operatorcode /comp=companyID /prog=appnam /Link=ABC

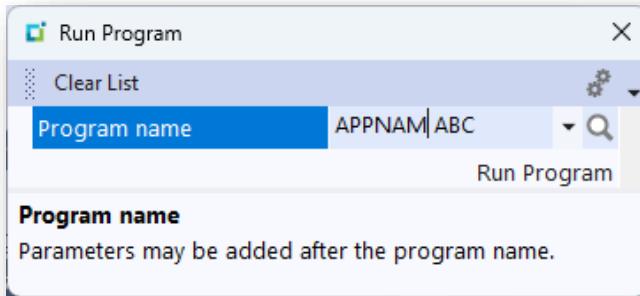
or this syntax if in client/server mode:

```
SYSPROClient22.exe /h /wcf=localhost:30110 /basedir=n
/oper=operatorcode /comp=companyID /prog=appnam /link=ABC
```

Note: Refer to [Command line parameters](#) in SYSPRO's online Help for more information on this.

The value in /Link will be returned in the **GetAppParameter** callout.

Similarly, you can pass this value to the application by appending the parameter to the program being called. Here's an example of doing this using the **RUN PROGRAM** facility in SYSPRO:



PASSING EXECUTION BACK TO THE CALLING PROGRAM

The **RaiseToolbarEvent** callout can be used to execute a toolbar button either in the current application or indeed the calling application. This can be especially useful when exiting an application and you need to make the calling application execute a toolbar button.

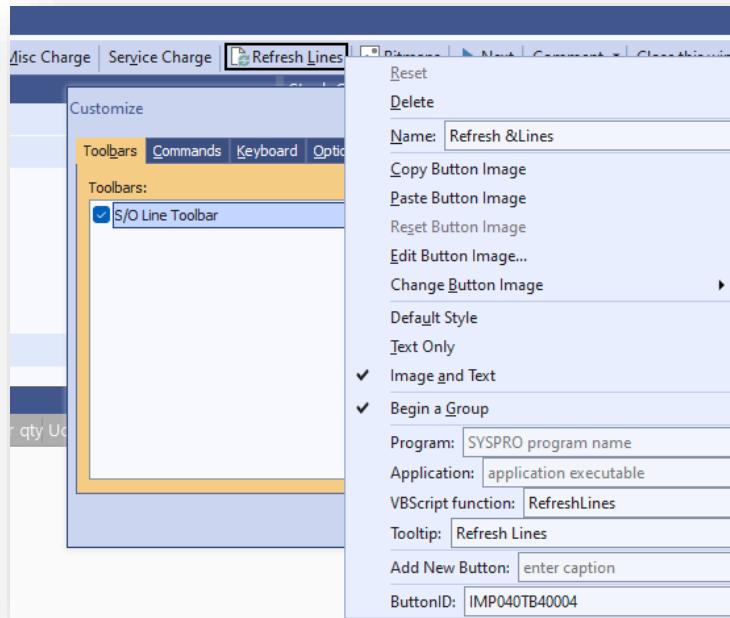
The syntax looks like this:

Raise an event on a toolbar	(RaiseToolbarEvent, "toolbarName", "button ID")	Execute a specified toolbar and button ID. This may be useful when exiting an application to execute a button on the calling application.
-----------------------------	---	---

You will need to know the name of the toolbar and the ID of the button on that toolbar. For example, if you have an application that is being called from Sales Order Entry (IMP040) and upon exiting you wish to cause the order lines to be 'refreshed' in IMP040 then you would use this syntax:

```
Function AppDesigner_OnExit()
    call CallsYSPROFunction(RaiseToolbarEvent, "IMP040TB", "40004")
End Function
```

To determine the toolbar name and button ID, just right-click on any toolbar in a SYSPRO program and select **Customize...**, then right-click on the target button and the property **ButtonID** shows the 8-character toolbar name followed by the button ID:



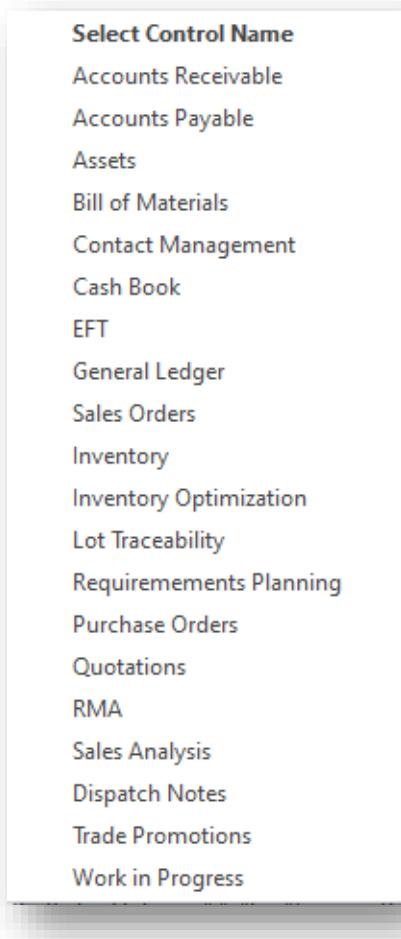
RETRIEVING SETUP OPTIONS

You may need to get access to setup options to help determine the behaviour of the application. While it is possible to call any of the Setup business objects from a script, this process is simplified with the callout **GetSetupOptions** in the script editor:

Show busy	(ShowBusy, " ", " ")	Changes the WAIT cursor to indicate the application is busy.
Show ready	(ShowReady, " ", " ")	Show the ARROW cursor to indicate the application is not busy.
Get Setup options	(GetSetupOptions, " ", "key")	Returns Setup options as a XML string.

Dockin pane callouts

Double-clicking on this item will display a list of modules:



Select the required module (e.g. Inventory) and a statement will be inserted into the script editor like this:

```
returnValue = CallSYSPROFunction(GetSetupOptions, " ", "INV")
```

On execution of this statement, **returnValue** will contain an XML string of options:

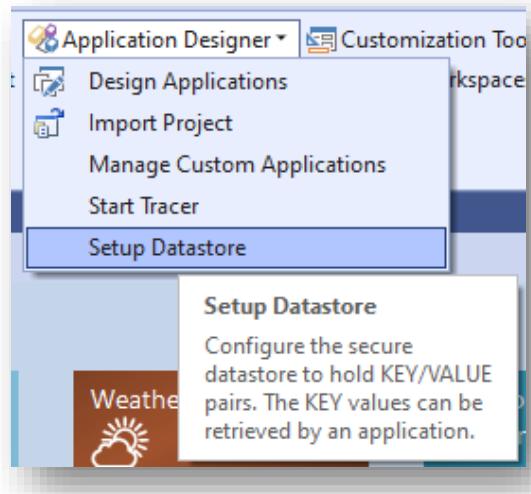


SETTING UP THE DATASTORE

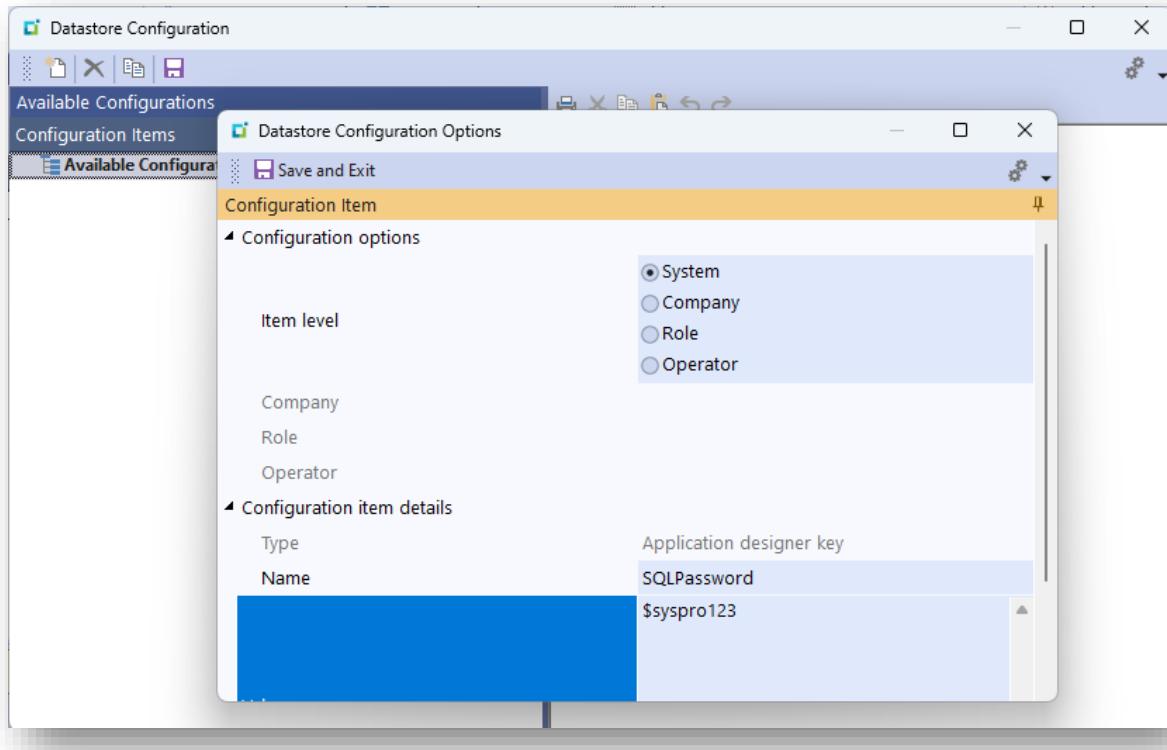
The Datastore can be used by a customer to store credentials or any other sensitive information in an encrypted database table. These take the form of a key/value named pair. The value of a given key can be retrieved by an application using the **GetDatastoreVariable** callout:

Get a datastore variable	(GetDatastoreVariable, " ", "key")	Returns the value of the KEY from the secure datastore. If the KEY is invalid then the value returned will begin with ERROR:. Typically this would be used to retrieve credentials.
--------------------------	------------------------------------	---

To setup the datastore in the first instance, click on the option **Setup Datastore** found in the Ribbon Bar:



The program ESPDST will be launched:



Click on the **New** button, select the level detail and then the **Name** (this is the KEY used in the **GetDatastoreVariable** callout) and **Value**.

The level detail follows the security logic of finding the value for a key based on the hierarchy of: Operator -> Role -> Company -> SYSTEM

For example, if 'SqlPassword' is saved company-wide for company '2' and operator 'Phil' is logged into company 2, then operator 'Phil' will receive this password.

However, if there is also a 'SqlPassword' store against the Operator 'Phil', then the value against 'Phil' will be returned and not the one against the Company.

Note: The values of KEYS are encrypted in the datastore in the database table [EspDatastore](#).

Understanding the Controls

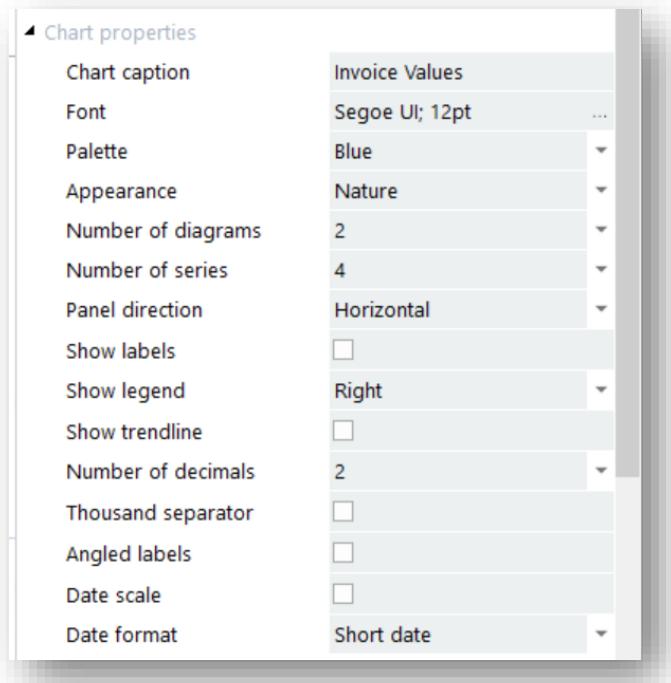
In this section we explore how to make best use of the various controls that make up an application.

CHART CONTROL

The chart control is a powerful, easy to use control with numerous ways to display data.

The look and feel options are defined when setting up the Chart pane in the Designer, leaving the developer to focus on constructing XML data to send to the chart.

MAIN CHART OPTIONS



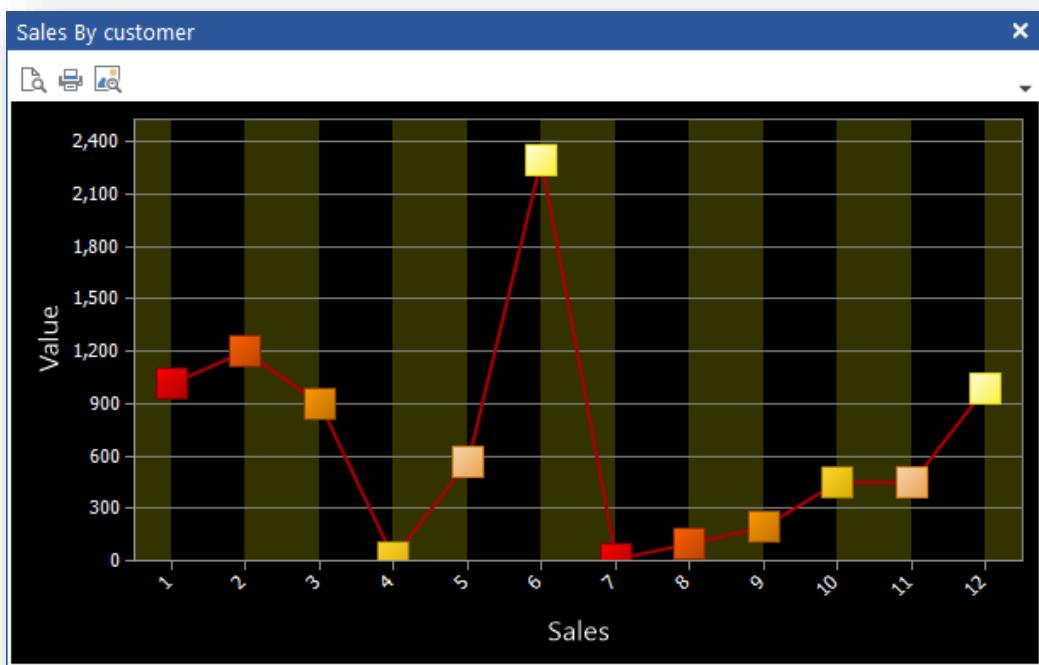
Using the control is straight forward. The Callout functions below explain the functionality available.

FEATURES OF THE CONTROL

There are several options to control the appearance of each diagram in a chart.

You may have up to 4 diagrams in one chart, and up to 4 series.

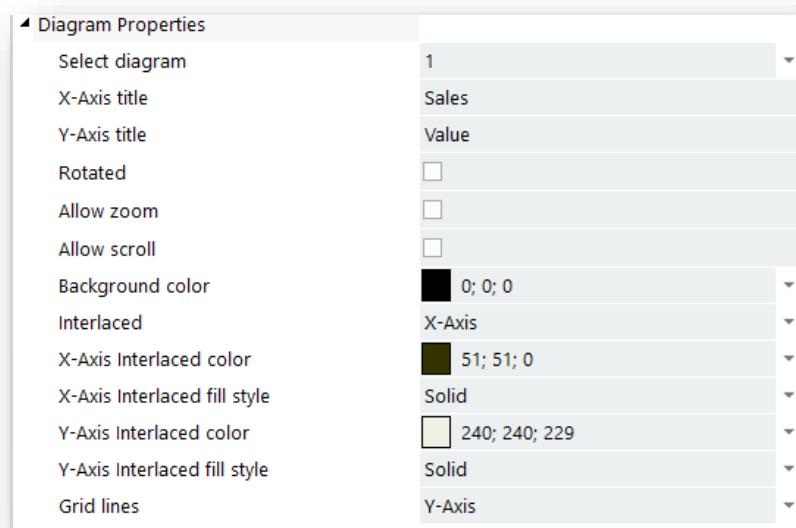
Each series can be in separate diagrams or grouped within a selected diagram.



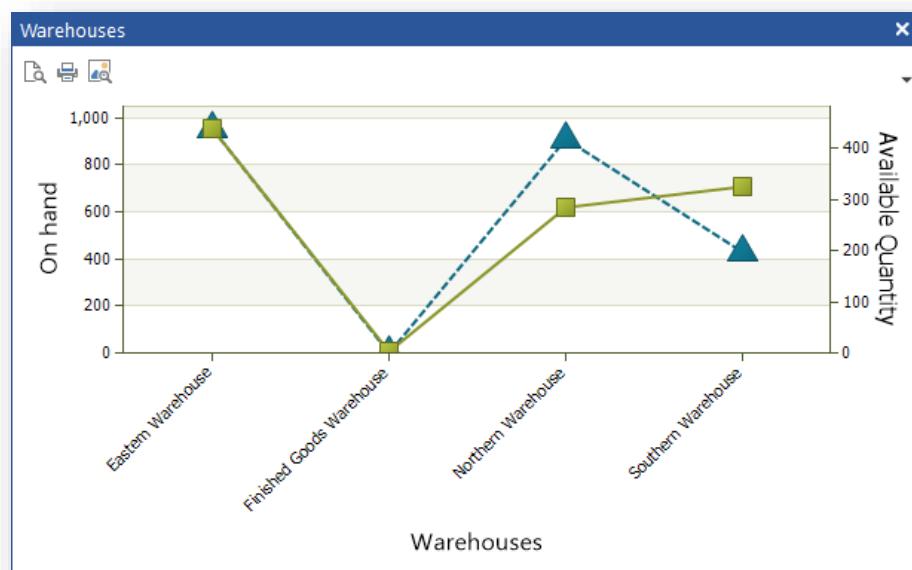
The properties of a diagram include:

- Interlaced colors
- Grid Lines
- Background colors
- Transparency option for Area and Bubble styles

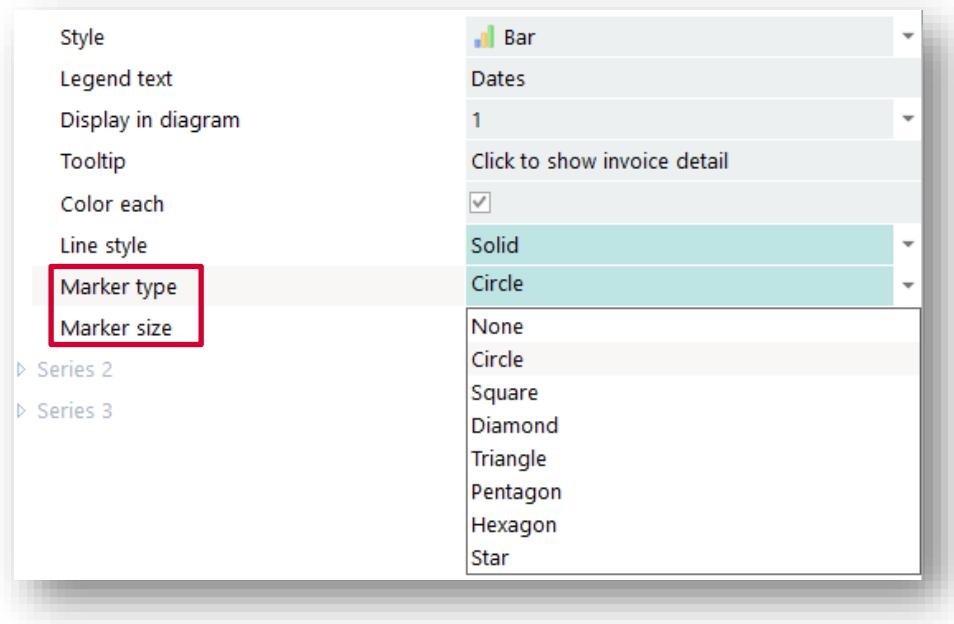
You can also rotate the diagram and allow the user to zoom into a chart area and then scroll the chart.



Additionally, you can show a **Secondary Y-Axis** for the chart. A secondary axis is useful when you need to display a series of data of values that greatly differ from each other, or you might use a secondary axis when displaying two series of data with different data type (i.e. price and quantity). In both cases you can plot each of the series on a secondary axis so that they can both be seen in the same view:



The properties for each series include Marker Type and Size:

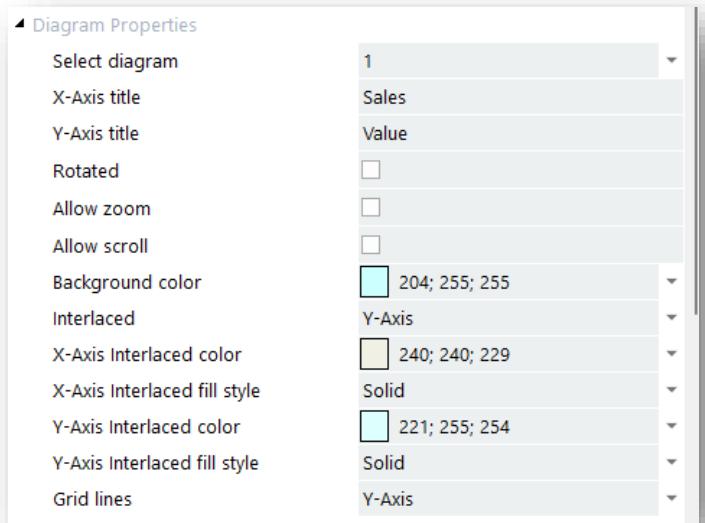


Each series can be displayed as a different chart style.

There is no limit to the amount of data (data points) that can be sent to a chart and a mini toolbar allows for print preview, printing and displaying in an image viewer.

For example:

DIAGRAM OPTIONS:



SERIES OPTIONS:

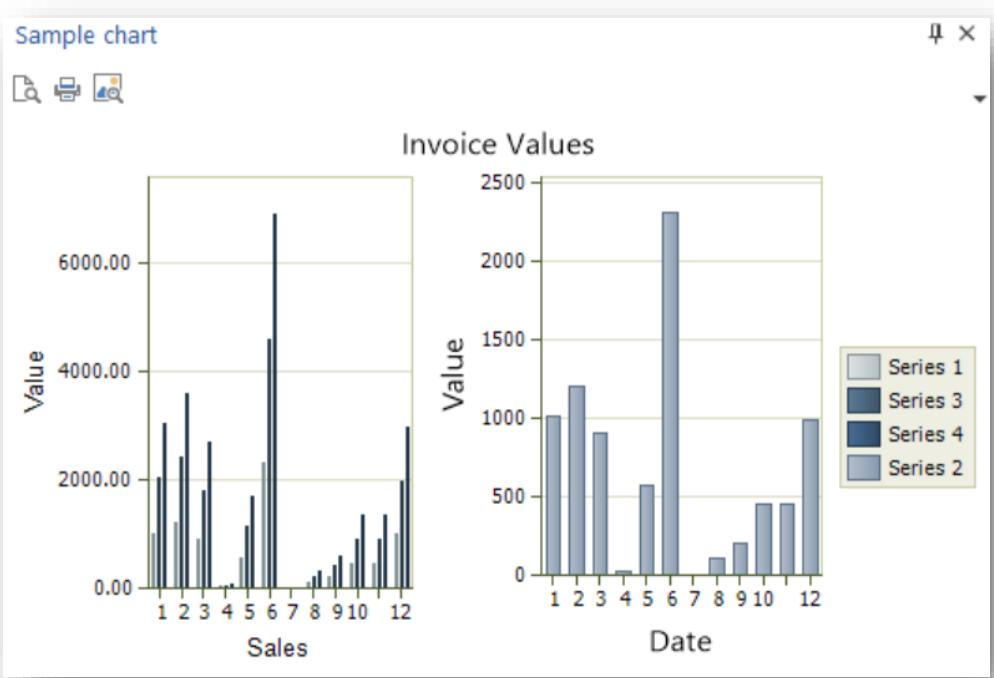
The screenshot shows the 'Series Options' dialog box. On the left, under 'Series 1', there are several configuration options:

- Style**: Set to Bar.
- Legend text**: Set to Dates.
- Display in diagram**: Set to 1.
- Tooltip**: Set to Click to show invoice detail.
- Color each**: Checked.
- Line style**: Set to Solid.
- Marker type**: Set to Circle.
- Marker size**: Set to 8.

On the right, a list of chart types is displayed:

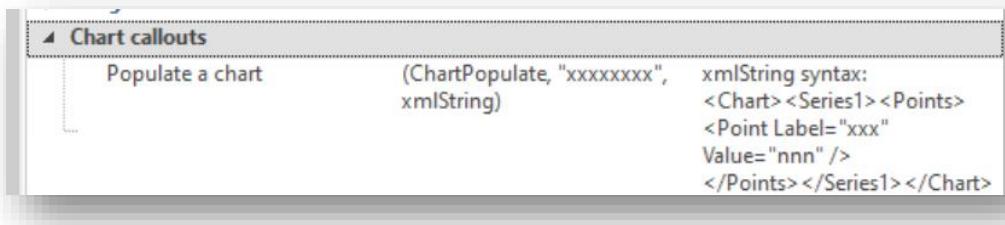
- Bar
- Bar
- Bubble
- Funnel
- Gantt
- Line
- Pie
- Point
- Pyramid
- Range bar
- Spline

OUTPUT:



CALLOUT FUNCTIONS

The Application Designer has the following callout functions for the Chart control:



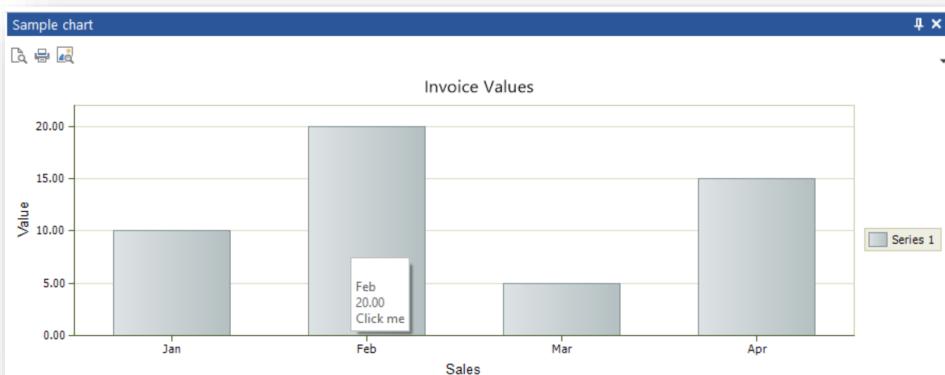
Therefore, code such as this:

```
9  Function PopulateChart  
10  
11  dim xmlString  
12  xmlString = "<Chart><Series1>" -  
13  & "<Points><Point Label='Jan' Value='10' /></Points>" -  
14  & "<Points><Point Label='Feb' Value='20' /></Points>" -  
15  & "<Points><Point Label='Mar' Value='5' /></Points>" -  
16  & "<Points><Point Label='Apr' Value='15' /></Points>" -  
17  & "</Series1></Chart>"  
18  Call CallSYSPROFunction(ChartPopulate, "ANDAPDCO", xmlString)  
19  
20  end function
```

will produce a chart like this:



Hovering over a column will show a tooltip of the x axis and y axis values, and if there is a tooltip defined in the series that will be shown as well:



EVENTS

Clicking on a column raises the **ChartClicked** event:

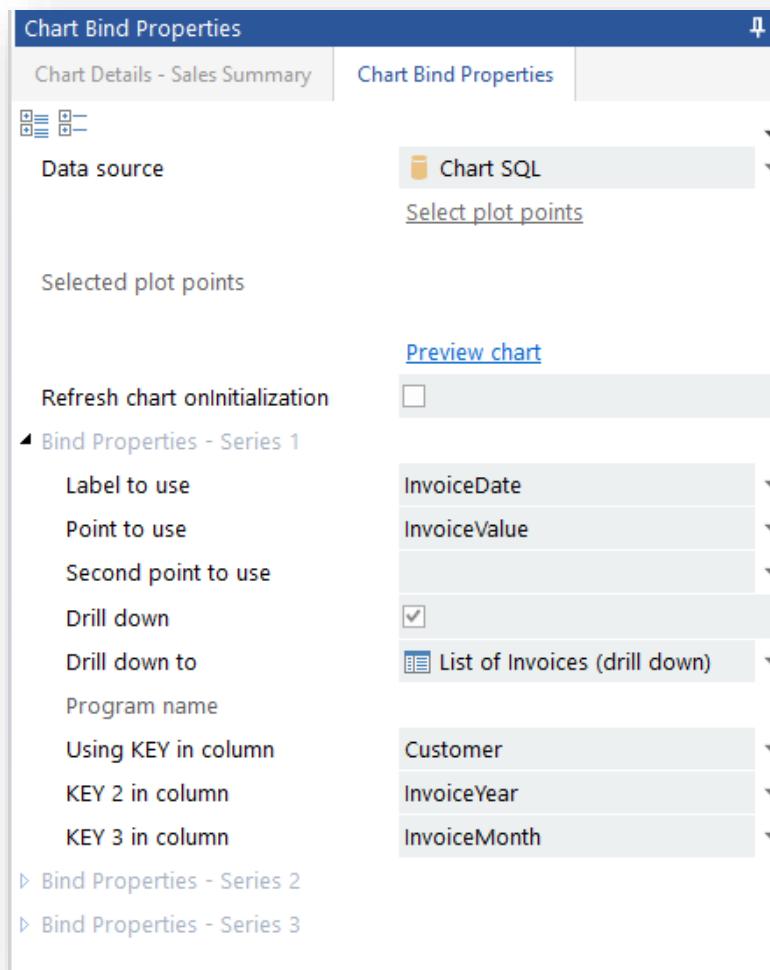
Fires when clicking on ChartClicked
a chart point.

Param1 contains the chart series, Param2 contains the chart point and Param3 contains the chart label.

DATA SOURCES AND CHARTS

You can bind data sources to any chart. Click on the **Chart Bind Properties** tab to see the available options and properties.

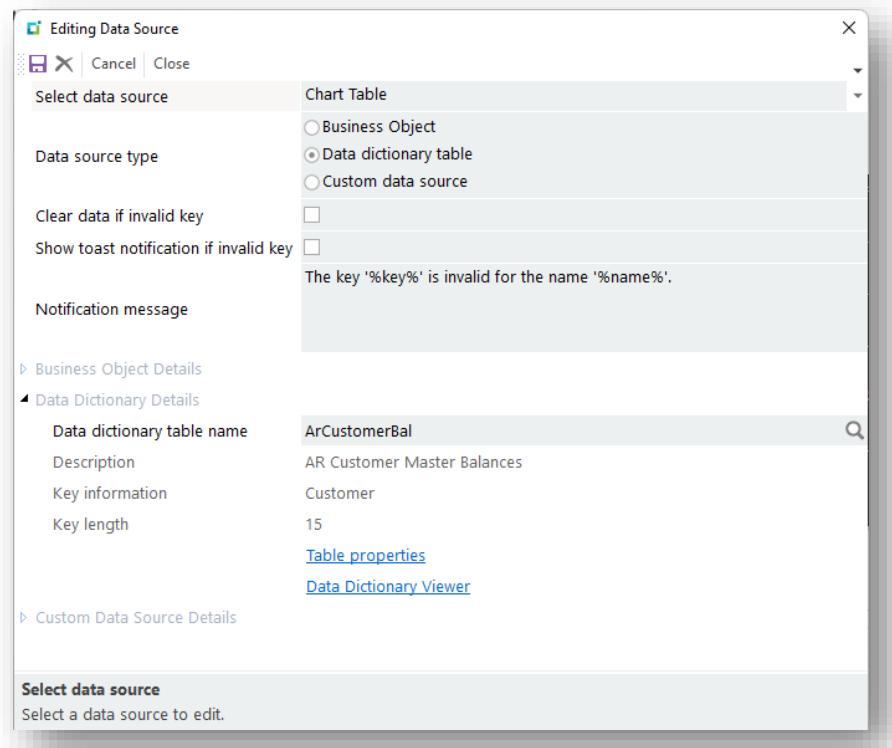
Typically, a chart's content will be refreshed when a specific data source is executed, either from being associated with an event (such as clicking on a toolbar button) or via code in a script. But you can also select the **Refresh chart OnInitialization** check box and the chart's data source will be executed as the application loads – note, however, that the data source would need to be constructed so that a 'key' is not required for it to execute correctly.



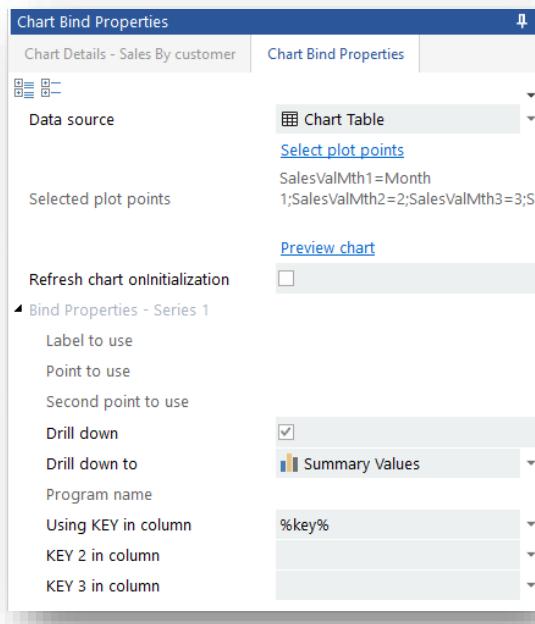
Select the data source to bind data for the chart. While only one data source can be selected per chart, you can select different elements from the data source to be shown in different series. How data is bound to a chart depends on the type of data source.

BINDING DATA TO A CHART USING A TABLE DATA SOURCE

A table data source can only provide chart points for the columns in the table. Here's an example of a data source based on the `ArCustomerBal` table.



The Chart Bind properties will look like this:



Click on **Select plot points** and the following window will appear:

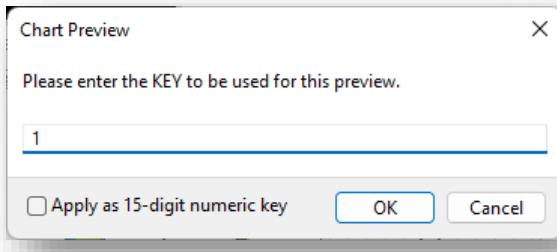
Chart Points from Data Source: Chart Table (ArCustomerBal)					
Select	Description	Data type	Column name	Length	Decimals
<input checked="" type="checkbox"/>	Month 1	Numeric	SalesValMth1	12	2
<input checked="" type="checkbox"/>	2	Numeric	SalesValMth2	12	2
<input checked="" type="checkbox"/>	3	Numeric	SalesValMth3	12	2
<input checked="" type="checkbox"/>	11	Numeric	SalesValMth11	12	2
<input checked="" type="checkbox"/>	12	Numeric	SalesValMth12	12	2
<input checked="" type="checkbox"/>	13	Numeric	SalesValMth13	12	2
<input checked="" type="checkbox"/>	14	Numeric	SalesValMth14	12	2
<input checked="" type="checkbox"/>	15	Numeric	SalesValMth15	12	2
<input checked="" type="checkbox"/>	16	Numeric	SalesValMth16	12	2
<input checked="" type="checkbox"/>	17	Numeric	SalesValMth17	12	2
<input checked="" type="checkbox"/>	18	Numeric	SalesValMth18	12	2
<input checked="" type="checkbox"/>	19	Numeric	SalesValMth19	12	2
<input checked="" type="checkbox"/>	20	Numeric	SalesValMth20	12	2
<input checked="" type="checkbox"/>	21	Numeric	SalesValMth21	12	2
<input checked="" type="checkbox"/>	22	Numeric	SalesValMth22	12	2
<input checked="" type="checkbox"/>	23	Numeric	SalesValMth23	12	2
<input checked="" type="checkbox"/>	24	Numeric	SalesValMth24	12	2
<input type="checkbox"/>	Average days to pay	Numeric	AveragePayDays	4	0
<input type="checkbox"/>	Balance forward 1	Numeric	BalBroughtFwd1	12	2
<input type="checkbox"/>	Balance forward 2	Numeric	BalBroughtFwd2	12	2
<input type="checkbox"/>	Balance forward 3	Numeric	BalBroughtFwd3	12	2
<input type="checkbox"/>	Current balance 1	Numeric	CurrentBalance1	12	2
<input type="checkbox"/>	Current balance 2	Numeric	CurrentBalance2	12	2
<input type="checkbox"/>	Current balance 3	Numeric	CurrentBalance3	12	2
<input type="checkbox"/>	Customer	Alpha	Customer	15	
<input type="checkbox"/>	MTD adjustments value 1...	Numeric	MtdAdjval1	12	2

The chart points are displayed for the selected items and in the sequence from left to right (as presented in the list).

First, you would select the columns that you want plotted and then drag the item up or down the list so that they are displayed in the correct sequence. You can drag more than one item by selecting multiple rows and then drag them up or down, and then click the **Select** check box while holding down the **CTRL** key.

Finally, you can also change the description of each item that will be shown as the chart label – just click in the **Description** column and type in the required new label.

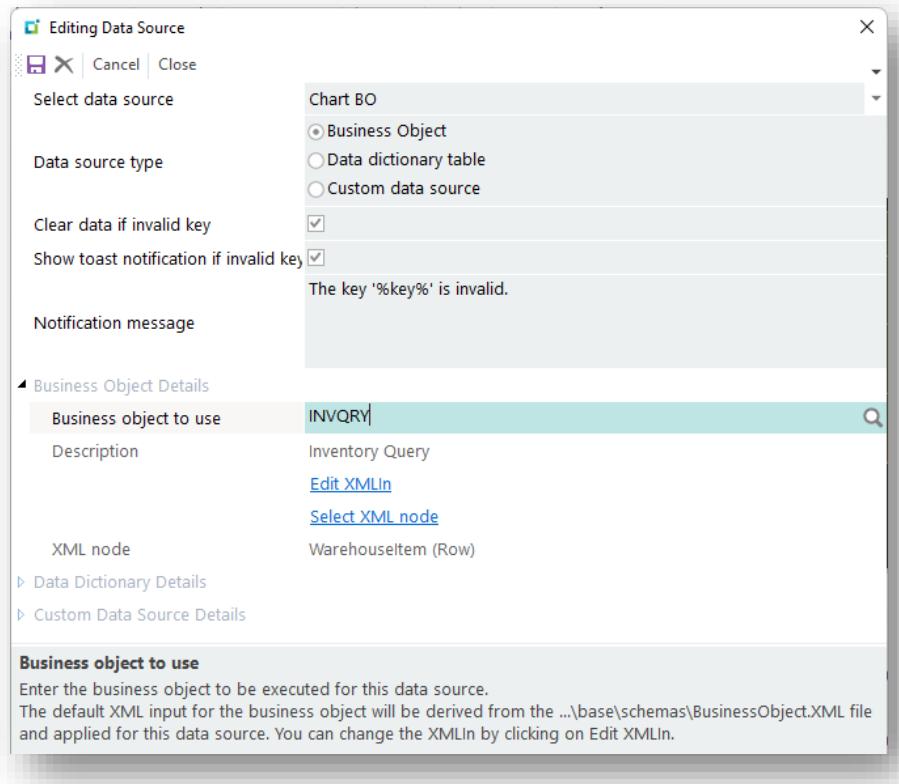
You can test out if your chart will work as expected by clicking on the **Preview chart** function – you will be prompted to enter a KEY for the preview to work:



BINDING DATA TO A CHART USING A BUSINESS OBJECT DATA SOURCE

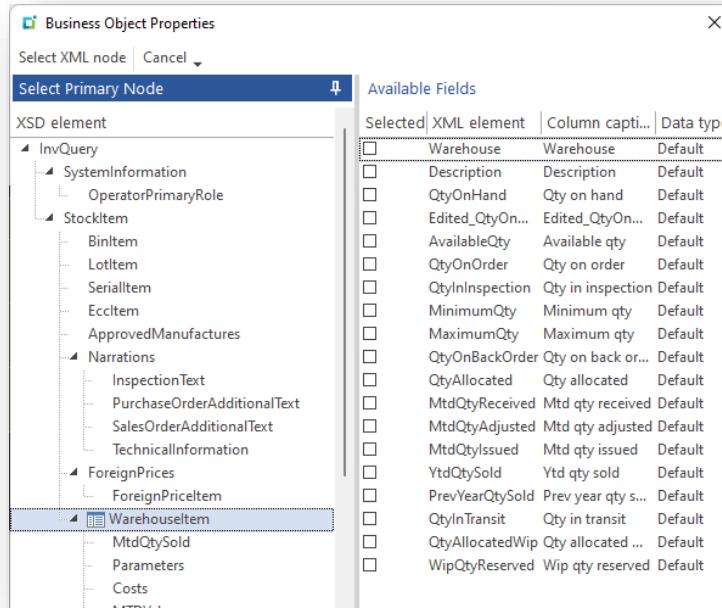
A business object data source can provide chart points for any columns provided from the selected XML node in the returned data.

In the following example, INVQRY is used to show all the warehouses in a chart:



Clicking on **Select XML node** allows you to select which XML node element to process.

In this case the **WarehouseItem** node is selected as this is a repeating row:



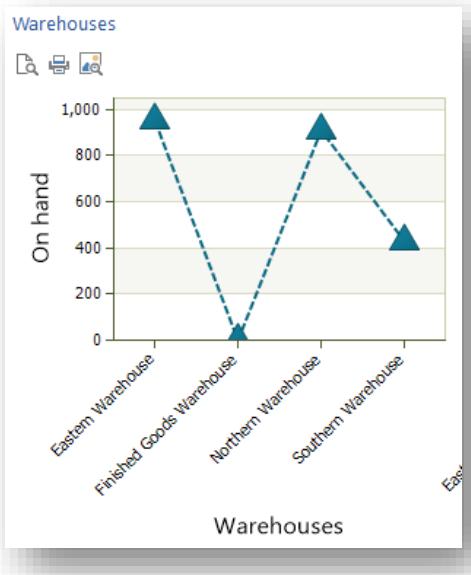
Within the Bind properties for the chart, the business object data source can be selected:

The screenshot shows the 'Chart Bind Properties' dialog box. At the top, there are tabs for 'Chart Details - Warehouses' and 'Chart Bind Properties'. Below these are sections for 'Data source' (set to 'Chart BO') and 'Selected plot points'. A 'Bind Properties - Series 1' section is expanded, showing a table with columns for 'Label to use' and 'Description'. The 'Description' column lists various XML elements from the 'WarehouseItem' node.

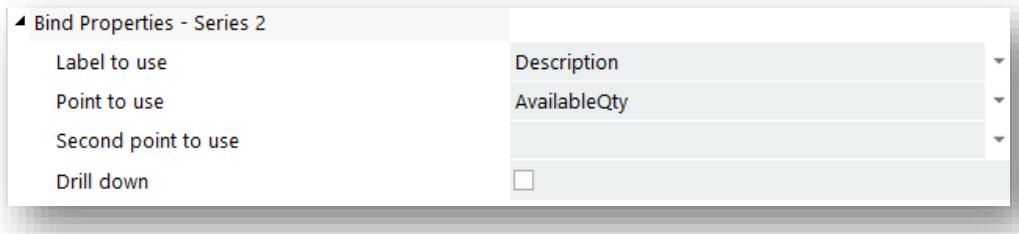
Label to use	Description
Point to use	Description
Second point to use	QtyOnHand
Drill down	Edited_QtyOnHand
Drill down to	AvailableQty
Program name	QtyOnOrder
Using KEY in column	QtyInInspection
KEY 2 in column	MinimumQty
KEY 3 in column	MaximumQty
	QtyOnBackOrder
	QtyAllocated

You can now select any available XML element from the **WarehouseItem** XML node for the **LABEL**, and then separately indicate which element value to show as the **POINT TO USE**. The values derived from these elements are used as the plot labels and points.

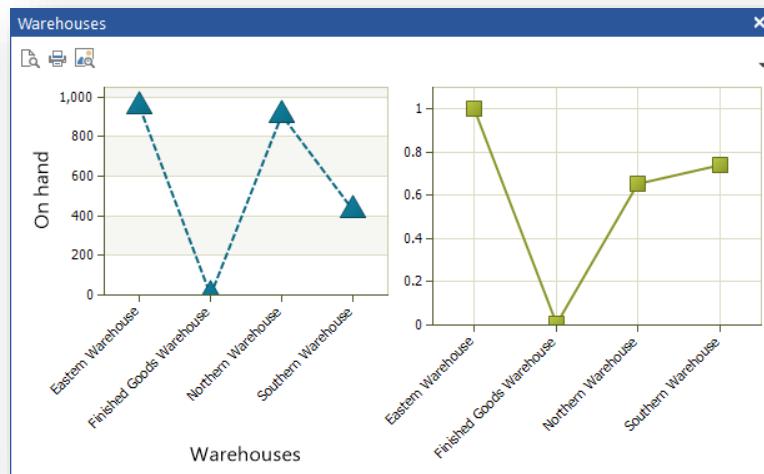
Clicking on the **Preview** hyperlink will show this:



You can of course select different elements to show in another series. So, while series 1 shows the Warehouse description and Qty on Hand, Series 2 can show the Warehouse description and Available Qty:



The result is this, assuming two diagrams and each series is in a different diagram:



BINDING DATA TO A CHART USING A CUSTOM DATA SOURCE

A custom data source can provide any data from any database. What's important to understand is that the chart can only plot points that are returned from the **SELECT** items in the SQL statement being executed.

Here's an example of a data source that aggregates invoice values for a given customer:

Editing Data Source

Cancel Close

Select data source	Chart SQL
Data source type	<input type="radio"/> Business Object <input type="radio"/> Data dictionary table <input checked="" type="radio"/> Custom data source
Clear data if invalid key	<input checked="" type="checkbox"/>
Show toast notification if invalid key	<input checked="" type="checkbox"/>
Notification message	The key '%key%' is invalid for the name '%name%'.
Business Object Details Data Dictionary Details Custom Data Source Details	
Database access method	OleDb Provider
Connection string to use	SYSPRO connection
Connection string	Provider=Microsoft OLE DB Provider for SQL Server;Data Source=\${ServerName};User Id=\${UserID};Password=\${Password};Database=\${CompanyDb}; Add Connection String Edit Connection string Delete Connection String Edit SQL statement <pre>select InvoiceYear, InvoiceMonth, Customer, InvoiceValue = SUM(CurrencyValue), NumberOfInvoices= COUNT(Invoice),InvoiceDate=format(Datefromparts(InvoiceYear,InvoiceMonth,'0000-00-00'),'yyyy-mm-dd') from ArInvoice where Customer = '%key%'</pre>
SQL statement	<input type="radio"/> Form <input type="radio"/> List <input checked="" type="radio"/> Chart
XML returned style	Test connection Test SQL statement
Select data source	

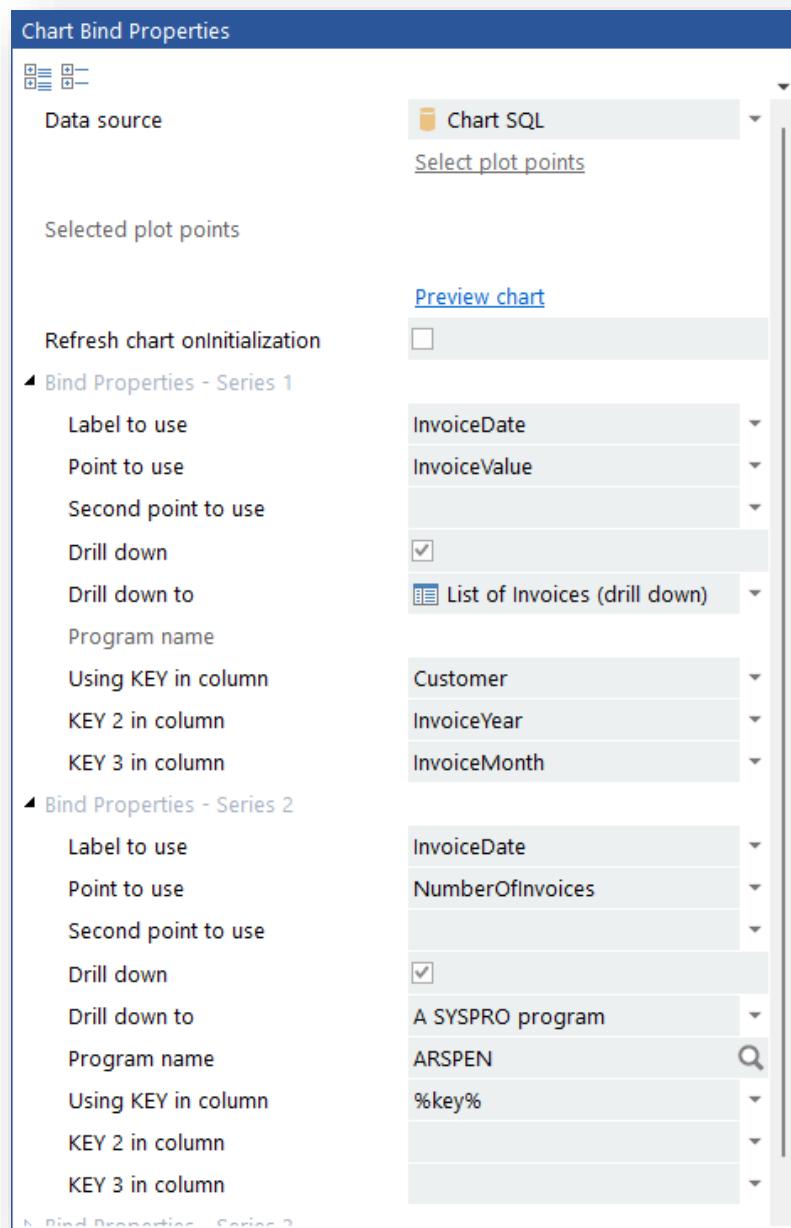
You will notice that the XML returned style is flagged as **Chart**. This ensures that the SQL statement will return Labels and Points in the XML string.

The SQL statement for this data source looks like this:

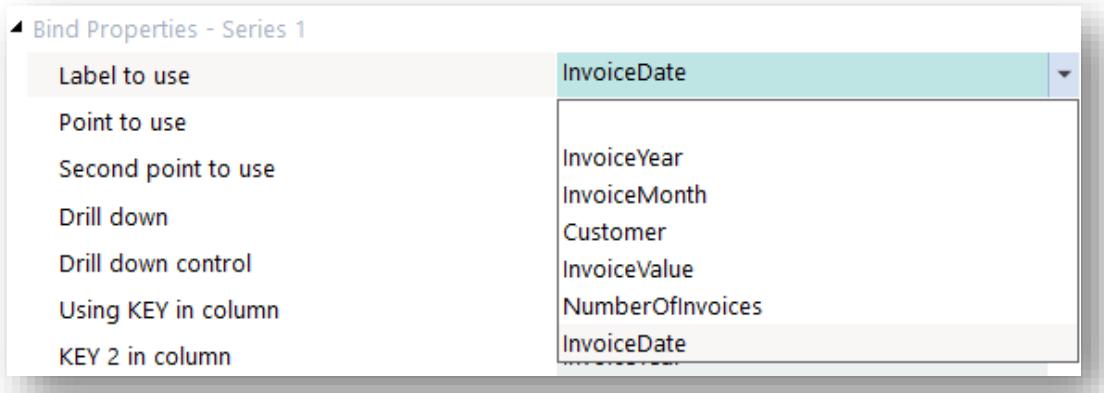
```
select InvoiceYear, InvoiceMonth, Customer, InvoiceValue =  
SUM(CurrencyValue),  
NumberOfInvoices=  
COUNT(Invoice), InvoiceDate=format(Datefromparts(InvoiceYear, Invoice  
Month, '01'), 'yyyy-MMM')  
from ArInvoice where Customer = '%key%'  
Group by InvoiceYear, InvoiceMonth, Customer  
order by InvoiceYear, InvoiceMonth
```

Notice the items in BOLD in the SELECT statement, as these will be the point values that can be selected to be displayed within the chart.

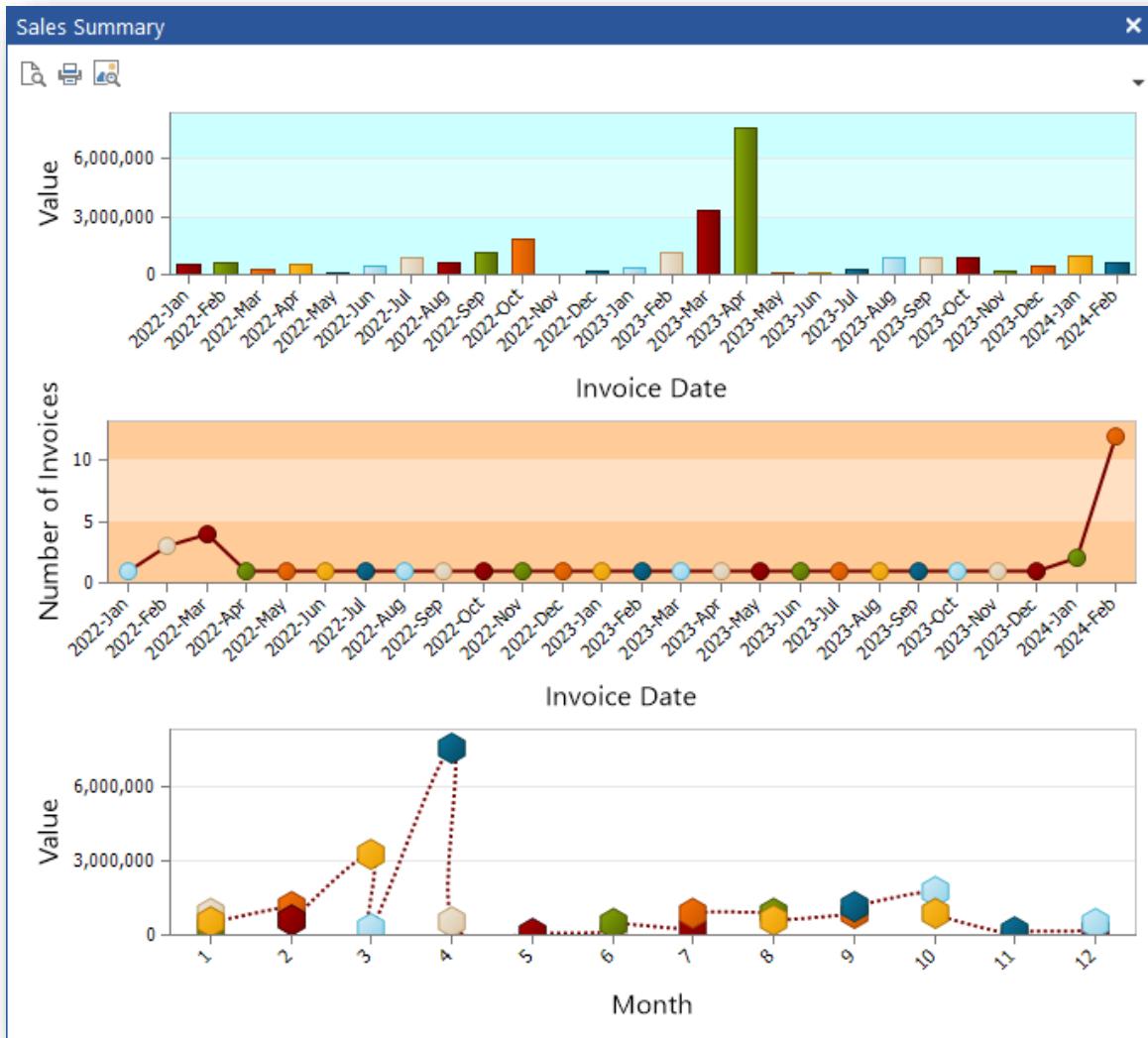
In the Bind properties for the chart, the SQL data source can be selected:



When you click on a LABEL or POINT TO USE dropdown list, you will see the available point values:



When previewing the chart, it will look like this (each series is being shown in a different diagram):



USING THE DRILL DOWN CAPABILITY FOR A CHART

You can, of course, use script to detect that a chart point has been clicked on and then provide your own drill down capabilities.

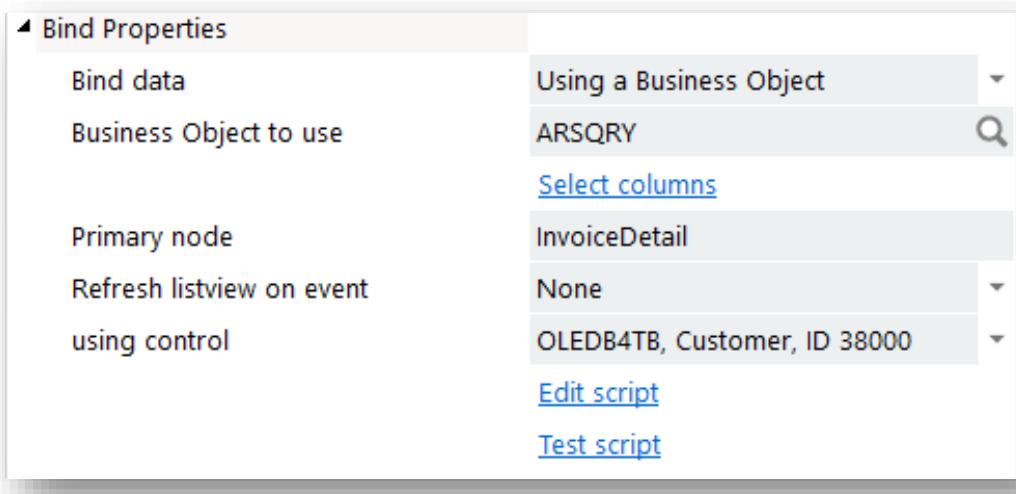
This section explains how you can use a data source to provide drill down capabilities without using any code.

Using our TABLE data source example, let's assume you want to see all the invoices for the customer that were used to execute the TABLE data source.

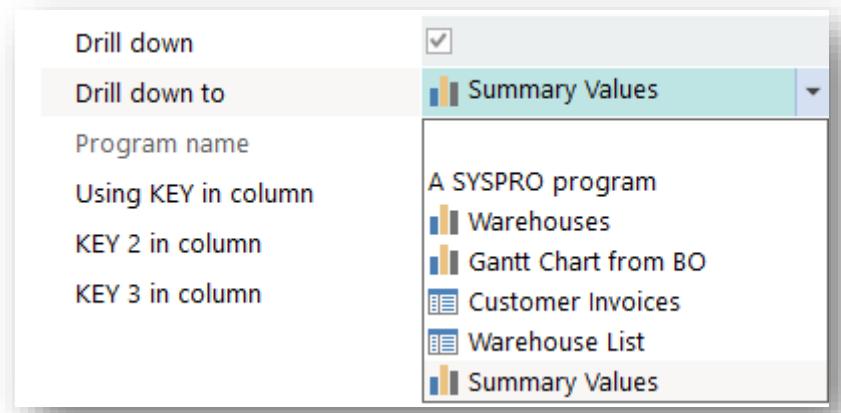
First, you will need to create a control to host a listview:



The BIND properties for the List view are simple enough, just using the ARSQRY business object and selecting the **InvoiceDetail** XML node to derive the list of invoices:

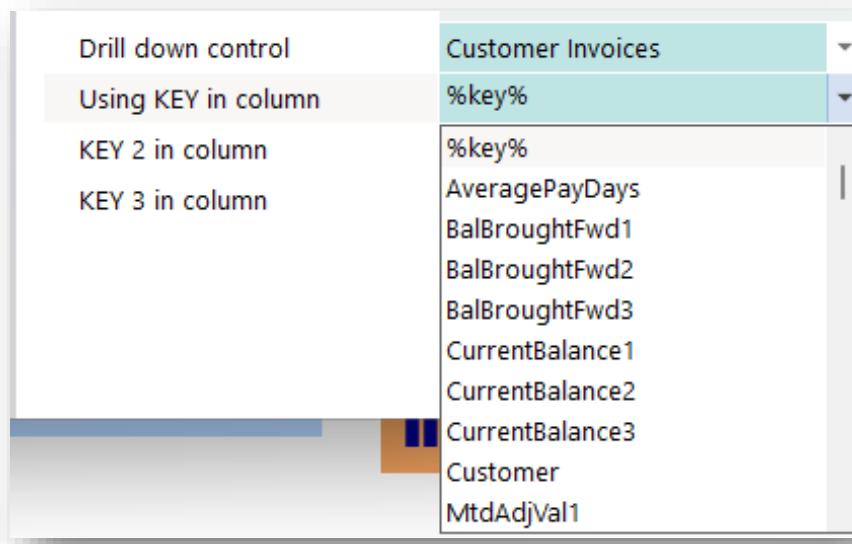


Next, in the BIND properties for Series 1 for the chart, the **Drill down** option is enabled and the **Customer Invoices** control is selected from the **Drill down control** list:



Note: Only controls that are listviews or charts will be shown in this list. Alternatively, you can select **A SYSPRO program** and then enter a program name to be executed for the drill down – the KEY value will be passed to the program.

Finally, you can select the column in the TABLE that will provide the **KEY** to be passed to data source in the Customer Invoices control:



You could have selected **Customer** as the obvious KEY to be used, but you can also use the variable **%key%** - as this indicates that whatever key was used to execute the chart data source will be the same value passed to the drill down control. In our case, this is the Customer key.

When you run the application and enter a Customer number, the chart could look like this:



Click on any data point and the invoice list will be displayed:

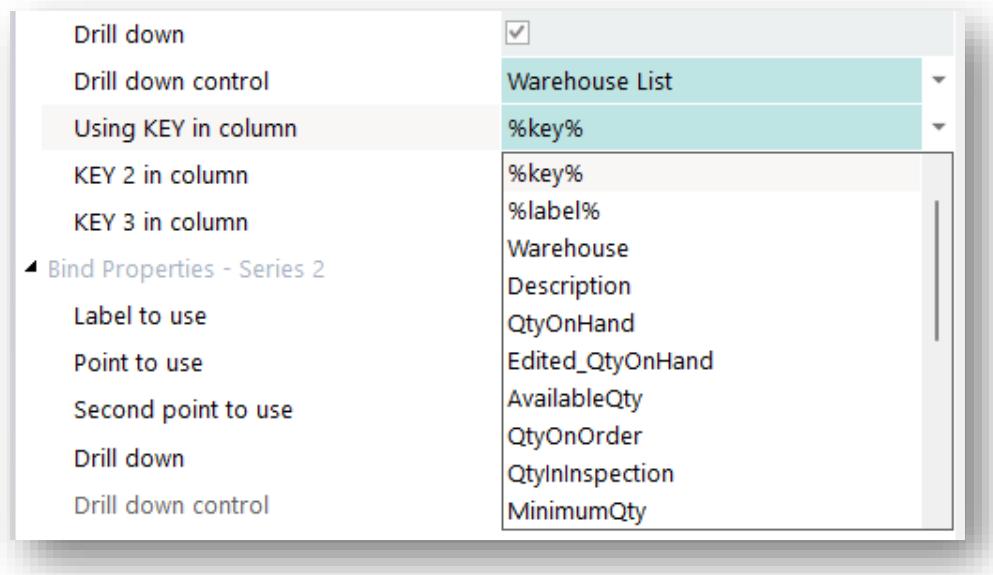
Customer Invoices		
Invoice	Invoice date	Invoice bal
100100	2023-01-18	5.67
100466	2023-02-09	448000.00
100476	2023-03-05	914000.00
100497	2023-04-01	584800.00
100509	2023-04-08	2110.00
100510	2023-04-08	784.00
100511	2023-04-08	1190.10
100512	2023-04-08	16493.75
100513	2023-04-08	2041.20
100514	2023-04-09	2100.00
102100	2022-02-16	12.99
800055	2023-04-08	-86.00
900036	2023-04-08	250.00
_CRD01	2023-03-31	-2500.00

Note that the drill down control will be automatically made visible (if hidden or closed) or shown as a modal dialog box (if the pane property **Show pane as dialog box** is checked).

DRILL DOWN FROM BUSINESS OBJECT OR CUSTOM DATA SOURCE

The same principle applies to a business object or custom data source.

One difference is that you can optionally select the variable %label%:



The %label% variable could be useful if the label for each data point is the code or key to execute another data source. For example, instead of showing the DESCRIPTION as the label, **Warehouse** could have been selected and in which case the value of the warehouse can be passed as the key to the drill down control.

Another difference is that you can select up to 3 KEY values to be passed to the drill down data source – this is only of use where the drill down data source is a Custom data source that can use these extra key values to execute a SQL statement. The 3 KEY values will be inserted into placeholders %key%, %key2% and %key3% .

Here's an example of such a SQL statement:

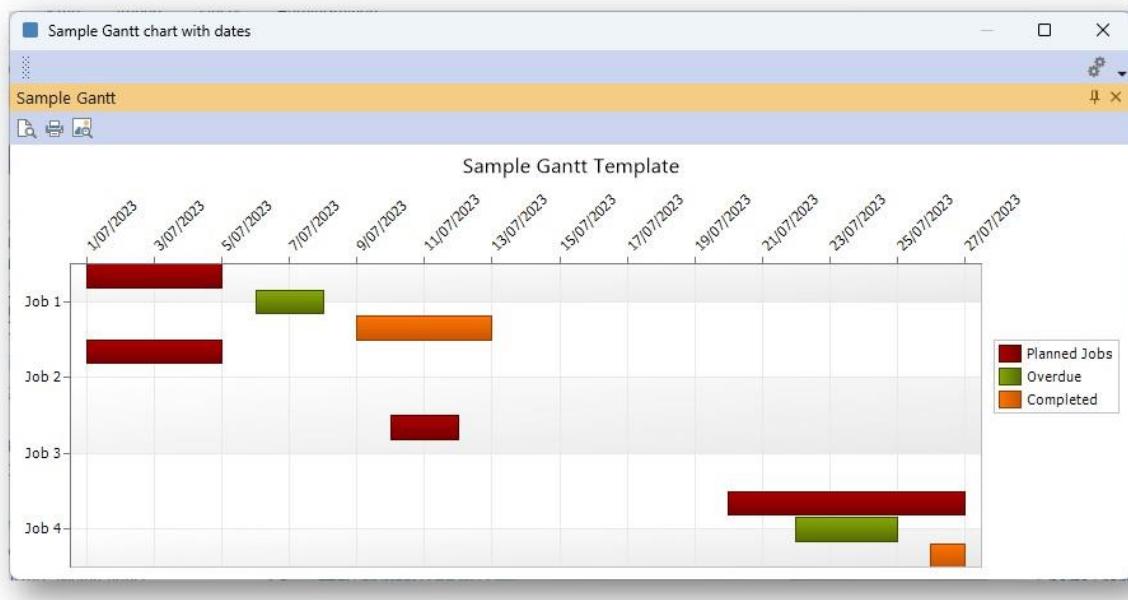
```
select InvoiceYear, InvoiceMonth, CurrencyValue  
from ArInvoice  
where Customer = '%key%'  
and InvoiceYear = '%key2%' and InvoiceMonth = '%key3%'
```

USING DATE VALUES AS CHART POINTS

You can use **DATES** as values in a Gantt chart. Dates must be presented in **CCYYMMDD** format, as in this example:

```
<Chart><Series1><Points><Point Label='Job 1' Value='20230701' Value2='20230705' />
```

And the chart will be shown like this:



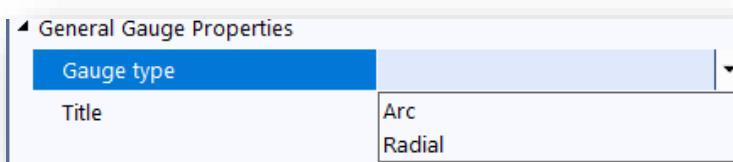
You can set these chart properties for dates to be used as values in a Gantt chart (a SHORT DATE format renders date labels as dd/mm/yyyy or mm/dd/yyyy depending on your date formatting):



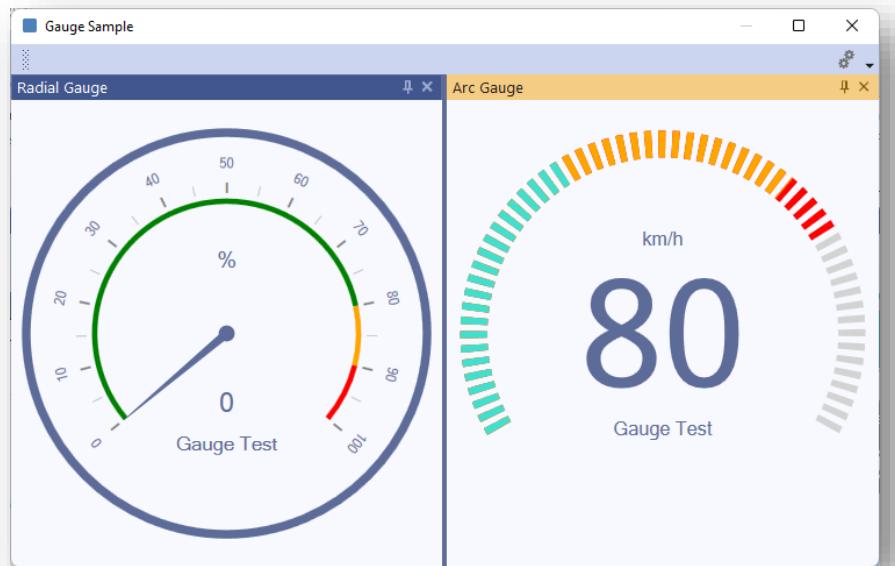
GAUGE CONTROL

You can add an **Arc** (displays a value range which is represented by an arc) and a **Radial** gauge (displays a metric as a percentage of the length of a circular scale, like a pie or donut chart with a single slice).

Apart from the normal Pane properties, you can select the type of Gauge to show:



The Radial gauge and Arc gauges are shown below:



The ARC gauge properties look like this:

Gauge Details - Arc

Pane Properties

General Gauge Properties

Gauge type: Arc

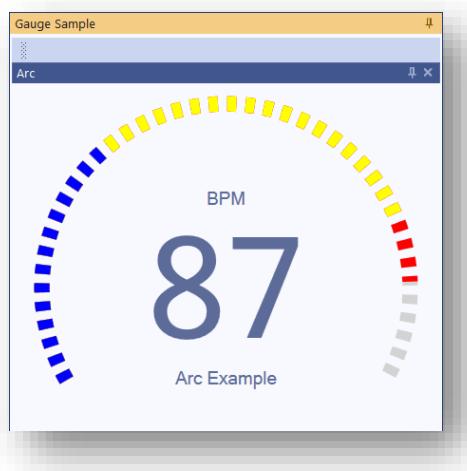
Title: Arc Example

Arc Properties

Setting	Value
Units caption	BPM
Arc thickness	12
Dash gap	0.60
Arc color	0; 0; 255
Arc angle	40-
Test value	87
Second arc color	255; 255; 0
Second arc angle	67
Third arc color	255; 0; 0
Third arc angle	130

[View documentation](#)

To render this gauge you can use the **Test value** slider to change the pointer value:



The Radial properties look like this:

Gauge Details - Radial Gauge

pane properties

General Gauge Properties

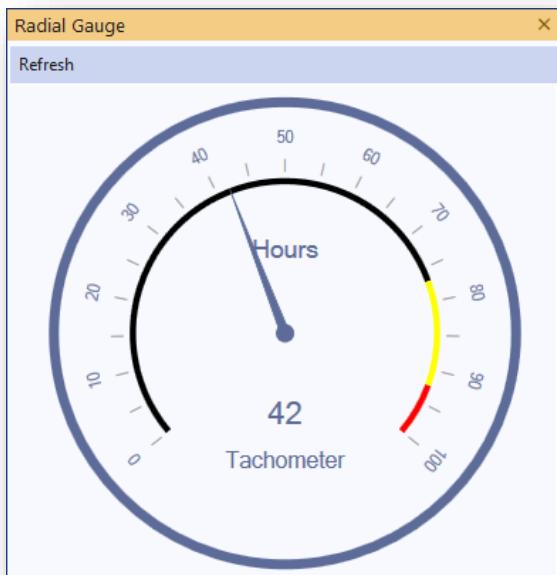
- Gauge type: Radial
- Title: Tachometer

Radial Properties

- Units caption: Hours
- Zero position: Left
- Divider increment: 5
- Scale increment: 10
- Maximum scale value: 100
- Label visible:
- Label position: Outside
- Test value: 42
- Range 1 visible:
Color: #000000
Start angle: 130
End angle: 70
- Range 2 visible:
Color: #FFFF00
Start angle: 70
End angle: 110
- Range 3 visible:
Color: #FF0000
Start angle: 110
End angle: 130

[View documentation](#)

To render this gauge you can use the **Test value** slider to change the pointer value:



All gauge properties are described in the Application Designer's Help panel.

After defining what you would like the gauge to look like, you can use the script callout **GaugeUpdate** to change the pointer value with a simple XML string:

▲ **Gauge callouts**

Update a gauge (GaugeUpdate, "xxxxxxxx", xmlString) xmlString syntax:<GaugeArc><Value>68...

LIST VIEW CONTROL

SOME BASICS

- A List view is populated by data from an XML string.
- The XML string must have a root node.
- Repeating rows of data should be contained in a repeating node.
- The List view control calls this repeating node “Primary node”.
- The data that populates a column is held in the text of child nodes of the repeating/primary node.
- When populating the data in a row, the List view control matches element/node names of the XML string with the column headings in the List view.

For example:

```
<Customer>
  <InvoiceDetail>
    <Customer>0000001</Customer>
    <Invoice>100466</Invoice>
    <InvoiceDate>2021-02-09</InvoiceDate>
    <CustomerPoNumber>BB801</CustomerPoNumber>
    <InvoiceBal>448,000.00</InvoiceBal>
  </InvoiceDetail>
  <InvoiceDetail>
    <Customer>0000001</Customer>
    <Invoice>100476</Invoice>
    <InvoiceDate>2021-03-05</InvoiceDate>
    <CustomerPoNumber>BB 805</CustomerPoNumber>
    <InvoiceBal>914,000.00</InvoiceBal>
  </InvoiceDetail>
  <InvoiceDetail>
    <Customer>0000001</Customer>
    <Invoice>100497</Invoice>
    <InvoiceDate>2021-04-01</InvoiceDate>
    <CustomerPoNumber>BB 809</CustomerPoNumber>
    <InvoiceBal>584,800.00</InvoiceBal>
  </InvoiceDetail>
  ...
</Customer>
```

In this example:

- The root node is **<Customer>**
- The repeating row node (Primary node) is **<InvoiceDetail>**
- The data comes from **<InvoiceDetail>** children
(E.g. **<CustomerPoNumber>BB801</CustomerPoNumber>**)

The XML above produces a List view like this:

Invoices from SYSPRO Business Object			
Invoice	Invoice date	Customer po number	Invoice bal
100466	2021-02-09	BB801	448,000.00
100476	2021-03-05	BB 805	914,000.00
100497	2021-04-01	BB 809	584,800.00
100509	2021-04-08	N900101-2004	2,110.00
100510	2021-04-08	N900101-2004	784.00
100511	2021-04-08	N900103-2005	1,190.10
100512	2021-04-08	N900805-2005	16,493.75
100513	2021-04-08	N900250-2004	2,041.20
100514	2021-04-09	567	2,100.00
800055	2021-04-08	CREDIT AUTHORISED	-86.00
900036	2021-04-08		250.00
_CRD01	2021-03-31	Unapplied cash	-2,500.00

The XML string can contain far more data than the List view needs. The extra data is simply ignored.

For example – using the SYSPRO business object ARSQRY, the XML looks like this:

```
<?xml version="1.0" encoding="Windows-1252"?>
<ARStatement Language='05' Language2='EN' CssStyle=' ' DecFormat='1' DateFormat='01'
Role='01' Version='8.0.034' OperatorPrimaryRole=' ' >
  <Header>
    <CompanyName>SYSPRO DEMONSTRATION</CompanyName>
    <CompAddress1 />
    <CompAddress2>P O Box 777</CompAddress2>
    <CompAddress3>Great Outdoors</CompAddress3>
    <CompTaxNo />
    <CompRegNo>1978/035534/04</CompRegNo>
    <Customer>0000001</Customer>
    ...
  </Header>
  <InvoiceDetail>
    <DocumentType>I</DocumentType>
    <Invoice>100466</Invoice>
    <InvoiceDate>2021-02-09</InvoiceDate>
    <DueDate>2021-03-11</DueDate>
    <CustomerPoNumber>BB801</CustomerPoNumber>
    <CurrencyValue>        448000.00</CurrencyValue>
    <InvoiceTerms>0</InvoiceTerms>
    ...
    <DunningInvoiceBal>0.00</DunningInvoiceBal>
    <DunningUser />
  </InvoiceDetail>
  <InvoiceDetail>
    <DocumentType>I</DocumentType>
    <Invoice>100476</Invoice>
    ...
    <Salesperson>100</Salesperson>
    <SalespersonName>Tony Dean</SalespersonName>
    <Area>N</Area>
    <AreaDesc>Northern Region</AreaDesc>
    <InvoiceAgeing>2022-04-02</InvoiceAgeing>
    ...
  
```

In this example:

- The root node is **<ARStatement>**
- The repeating row node is **<InvoiceDetail>**

Any data in child nodes that do not match the column headings is ignored. So, this XML will produce the same listview as the previous example, despite it holding much more data.

The sample App SAMDTA is a good example of how different methods of retrieving data can create an XML string and use the same listview for the different data sources.

CONSIDERATIONS

- The Primary node, column headings and XML node names are case sensitive.
- If the List view does not populate at all (i.e. it remains blank) then it is likely that it cannot find its primary node.
Check spelling and case sensitivity.
- If a column is missing data, then it means the listview cannot match the heading to any node in the xml.
Check spelling and case sensitivity.

ADVANCED FORMATTING

The XML string can be a set of simple XML elements with values but can also be constructed with various properties such as making a cell editable (or not) and changing cell colors.

The various properties that can be assigned to a cell value include:

Property	Entry
IsReadOnly	True or False
Value	The value of the cell
Background	A named color
Foreground	A named color
Tooltip	The tooltip for the cell
IsBold	True or False
IsItalic	True or False
XAMLCode	A named XAML theme to be applied to the cell

Here is some example code:

```
dim xmlString
xmlString = "<Customers><Customer>" &_
"<InvoiceDetail>" &_
"<SalesOrder IsReadOnly='true' Value='837' Background='Blue'" &_
Foreground='white' IsItalic='true' />" &_
"<CustomerPoNumber IsReadOnly='true' Value='abc900' Tooltip='This is a" &_
po number' />" &_
"<InvoiceDate Value='19890110' />" &_
"</InvoiceDetail>" &_
"<InvoiceDetail>" &_
"<SalesOrder IsReadOnly='true' Value='840' Background='Blue'" &_
Foreground='white' IsItalic='true' />" &_
"<CustomerPoNumber IsReadOnly='true' Value='c:\images\chateau.jpg'" &_
XAMLCode='Thumbnail' />" &_
"<InvoiceDate Value='20221031' />" &_
"</InvoiceDetail>" &_
"</Customer></Customers>"
call CallSYSPROFunction(ListviewClearAddRecords, "DTS004L0", xmlString)
```

This code will render as follows:

Invoices			
Sales order	Invoice	Invoice date	Customer number
000837		10 Jan 1989	abc900
000840		31 Oct 2022	

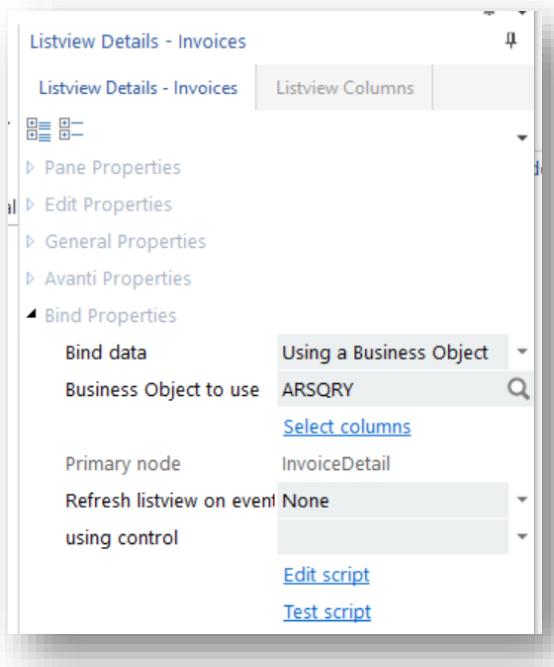
Note: Named colors may be:

- Black
- Blue
- Cyan
- White
- Green
- Red
- Magenta
- Yellow
- Gray

These may also be preceded by the word **Dark** or **Light** (e.g. Dark Cyan or Light Red).

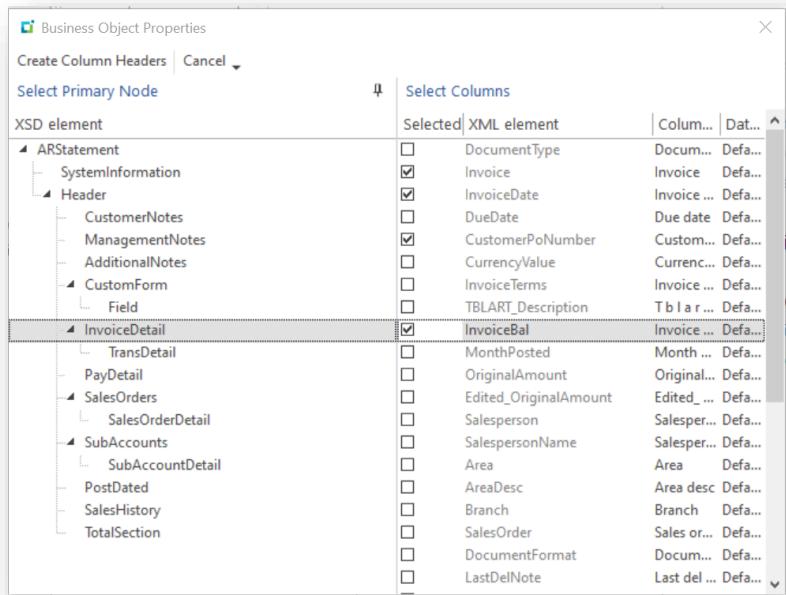
SETTING UP THE LIST VIEW CONTROL

Designing the listview and binding it to a data source is done in the **Bind Properties** of the List view:

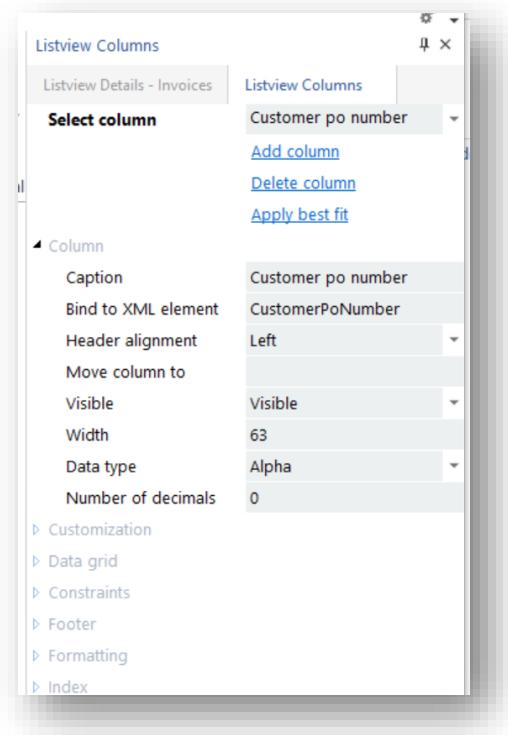


The key tasks to perform here are to set the “Primary node” and set the column headings. The **Bind data** option and the **Business Object to Use** help the user do this. But note, they have nothing to do with the actual operation of the listview when the App runs. They are used for design and configuration purposes only.

If **Bind data** is set to **Using a Business Object** and the **Business Object to use** is set, then the **Select Columns** option can be clicked: this launches a wizard that helps identify the Primary node and the columns required.



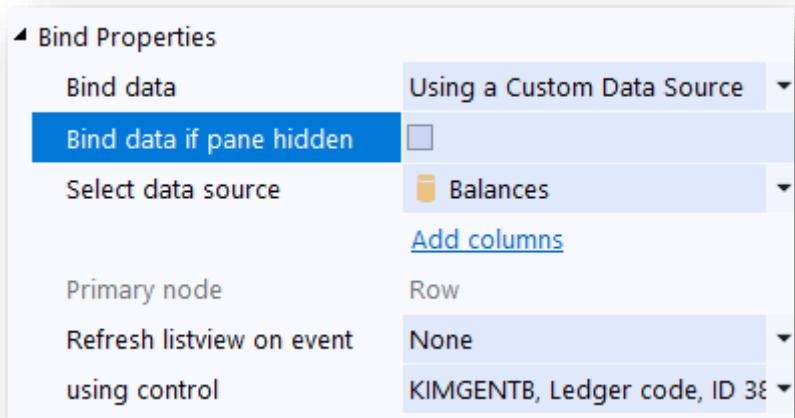
Columns can be added, deleted, and have properties set by selecting the **Listview Columns** pane.



Note that the column heading can have a different heading to the **Binding to the XML** element. In this case the listview will be looking for an XML element called "**CustomerPoNumber**" while the heading will display "Customer po number".

Please read the section on Data Sources to understand how to bind data to a listview using a Custom or Business object data source.

If you have setup to bind data for a listview then by default the execution of the bind data will always occur. However, if the listview is in a docking pane that is either closed or hidden then you can opt to prevent the bind data from executing. This may improve performance since if a pane is hidden the data is not going to be retrieved to populate the grid. To implement this, just check the option **Bind data if pane hidden**:

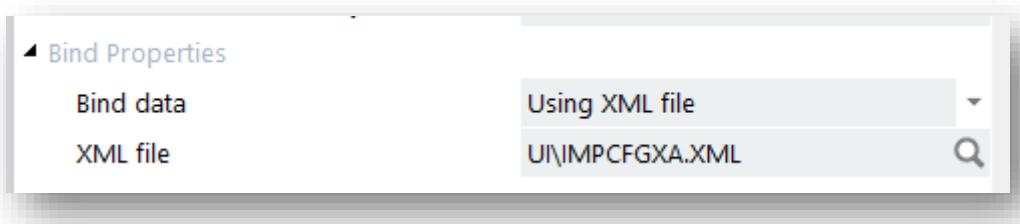


Considerations:

- This will also apply in Avanti, in that if a grid is not visible in a web view then the grid will not be populated with data.
- If a pane is hidden then the contents of the listview will be cleared. This is to ensure that if you subsequently make the pane visible that any previous data has been cleared out.

USING A PRE-DEFINED XML STRING

You can bind data to a listview directly using a specially formatted XML file. The BIND DATA option should be defined as **Using XML file**. You then define the XML file that you wish to show in the listview. This would be useful if you have a pre-defined listview structure to show. The disadvantage of this technique is that the XML file would have to be deployed to the target machine.



The XML file must be in a specific format – you can look at the file ...\\base\\UI\\IMPCFGXA.XML for an example of the XML structure.

When the application loads, the contents of the XML file will be automatically loaded.

Here's what this particular XML file will look like:

Category	Form name
▷ Configuration	
▷ Preferences	
▷ Financial Periods	
▷ Tax	
▷ History	
▷ User-defined Fields	
▷ Keys	
▷ Company	
▷ General Ledger Integration	
▷ System Setup	

SENDING THE XML STRING TO THE LIST VIEW

Irrespective of the way the XML string is derived, sending the data to the listview in the VBScript is the same.

For example:

```
call CallSYSPROFunction(ListviewClearAddRecords, "SAMDTAL0", xmldata)
```

where the **xmldata** variable holds the XML string.

This callout function will clear the listview grid and pass the xml document to the listview.

The listview identifies its primary node and matches the data elements with the column headings and displays the data.

You can update an existing listview using the callout **ListviewUpdateRecords**. This callout updates specific records in the grid using XML format. The XML string must contain the **RowIndex** (the row to update in the grid) and may contain new values, icons and/or tooltips.

The XML element names are used to match to the grid's column captions (for example, an XML name of **SalesOrder** will be used to update a column whose caption is **Sales order**).

If this is not precise enough for matching, you can instead provide the specific column index using the attribute **Column='nnn'**.

Update records as XML
 (ListviewUpdateRecords,
 "xxxxxxxx", xmlString)

Updates records in XML format. The XML element names must be column captions, or you can specify a specific Column.
Example input: <Records><Record
RowIndex='1'><SalesOrder Column='nn' Icon='nnn'
Tooltip='new tooltip'
>000793</SalesOrder></Record></Records>

The XML String looks like this:

```
<Records><Record RowIndex='1'><Invoice Icon='223' Tooltip='Important!'  
Column='1'>123456789</Invoice></Record></Records>
```

And finally, you can execute the listview's data source (to refresh the data) using the script callout **ListviewRefreshBindData**:

Refreshes the listview for Bind Data

(ListviewRefreshBindData, "xxxxxxxx",
"key")

Executes the data source associated with the Bind Data properties for the listview.

MANIPULATING THE LISTVIEW IN THE DESIGNER

The listview control has several **Events** that can be captured by the VBScript.

These are indicated here:

Callout function	Syntax	Remarks
Fires when right-click on a listview row.	ListRowRightClick	Param1 contains the row with cell values separated by a TAB character. Param 2 contains the row index.
Fires when double-click on a listview row.	ListRowDoubleClick	Param1 contains the row with cell values separated by a TAB character. Param 2 contains the row index.
Fires when a listview cell value changes.	ListCellChanged	Param1 contains the row with cell values separated by a TAB character. Param 2 contains the row index. Param 3 contains the column index (0-based).
Fires when clicking on a listview hyperlink.	ListHyperlinkClicked	Param1 contains the row with cell values separated by a TAB character. Param 2 contains the row index. Param 3 contains the column index (0-based).
Fires when a listview check box is changed.	ListCheckBoxChanged	Param1 contains the row with cell values separated by a TAB character. Param 2 contains the row index. Param 3 contains the column index (0-based).
Fires when the DEL key has been pressed on a listview row.	ListDELPressed	

Using a combination of events and callouts, it is possible to add, delete, and update rows within the listview.

There is an extensive list of script Callout Functions:

List view callouts		
Add records to a list view	(ListviewAddRecords, "xxxxxxxx", xmlString)	The XML string can be generated from a business object, or can contain XML elements incorporating properties. Example: <SalesOrder IsReadOnly='true' Value='840' Background='Blue' Foreground='white' Italic='true' IsBold='true'/> Colors may be black, blue, brown, cyan, darkblue, darkcyan, darkgray, darkgreen, darkmagenta, darkred, gray, green, lightblue, lightcyan, lightgreen, lightmagenta, lightred, red, white, yellow. The syntax for the XML string is as for ListviewAddRecords.
Clear all rows and then add records to a list view	(ListviewClearAddRecords, "xxxxxxxx", xmlString)	
Refreshes the listview for Bind Data	(ListviewRefreshBindData, "xxxxxxxx", "key")	Executes the data source associated with the Bind Data properties for the listview.
Clear all rows in a list view	(ListviewClearGrid, "xxxxxxxx", " ")	Returns a 2-dimensional array of rows and columns.
Get all records in a list view	(ListviewGetRecords, "xxxxxxxx", " ")	Returns a 2-dimensional array of rows and columns. Returns only records where the check box in the specified column is marked as 'checked'.
Get all 'checked' records in a list view	(ListviewGetCheckedRecords, "xxxxxxxx", "nnn")	Returns a 2-dimensional array of rows and columns. Returns only records that are selected.
Get all 'selected' records in a list view	(ListviewGetSelectedRecords, "xxxxxxxx", " ")	Note that this callout will not work in Avanti.
Get all 'matched' records in a list view	(ListviewGetMatchedRecords, "xxxxxxxx", "textToMatch nnn")	Returns a 2-dimensional array of rows and columns. Returns only records where a cell matches the specified text in a specified column.
Get all 'sorted' records in a list view	(ListviewGetSortedRecords, "xxxxxxxx", " ")	Returns a 2-dimensional array of rows and columns. Returns the records in the current sorted order.
Get all 'changed' records in a list view	(ListviewGetChangedRecords, "xxxxxxxx", " ")	Returns a 2-dimensional array of rows and columns. Returns only records that have changed.
Get all child records in a list view	(ListviewGetChildRecords, "xxxxxxxx", "nnnn")	Returns a 2-dimensional array of rows and columns. Returns the child records of a specified parent row.
Get records as XML	(ListviewGetRecordsAsXML, "xxxxxxxx", " ")	Returns the records in XML format. The column captions are used as the XML element names. Example output: <Records><Record RowIndex="1"><SalesOrder>000793</SalesOrder></Record></Records>
Update records as XML	(ListviewUpdateRecords, "xxxxxxxx", xmlString)	Updates records in XML format. The XML element names must be column captions, or you can specify a specific Column. Example input: <Records><Record RowIndex='1'><SalesOrder Column='nn' Icon='nnn' Tooltip='new tooltip'>000793</SalesOrder></Record></Records>
Get selected row	(ListviewGetSelectedRow, "xxxxxxxx", " ")	Returns the currently selected row with the row index followed by a TAB character and then all cell values separated by a TAB character.
Checks if the grid changed	(ListviewCheckForChanges, "xxxxxxxx", " ")	Returns a numeric value. 0 No changes 1 Records have changed 2 Records have been inserted
Update a row	(ListviewUpdateRow, "xxxxxxxx", "nnnn cell values")	Updates a row using a row index and cell values separated by a TAB character.
Update a cell in 'selected' row	(ListviewUpdateCell, "xxxxxxxx", "nnn text")	Updates the cell in column nnn with the specified text for the currently selected row.
Get specific row	(ListviewGetRow, "xxxxxxxx", "nnnn")	Returns the row at row index nnnn with cell values separated by a TAB character.
Delete selected row	(ListviewDeleteSelectedRow, "xxxxxxxx", " ")	Deletes the currently selected row(s).
Delete row at	(ListviewDeleteRowAt, "xxxxxxxx", "nnnn")	Deletes the row at row index nnnn.
Disable listview	(ListviewDisable, "xxxxxxxx", " ")	Disables the listview.
Enable listview	(ListviewEnable, "xxxxxxxx", " ")	Enables the listview.
Make columns editable	(ListviewCellsEditable, "xxxxxxxx", "nnnn list")	Makes one or more columns editable for a specific row. The list contains column numbers separated by a comma. Example: CallsYSRFunction(ListviewCellsEditable, "CUS001L0", "2 4,5") This will make columns 4 and 5 in row 2 editable.
Make columns not editable	(ListviewCellsNotEditable, "xxxxxxxx", "nnnn list")	Makes one or more columns not editable for a specific row. Same syntax as for ListviewCellsEditable.
Make all columns editable	(ListviewAllCellsEditable, "xxxxxxxx", "nnnn")	Makes all columns editable for a specific row.
Make all columns not editable	(ListviewAllCellsNotEditable, "xxxxxxxx", "nnnn")	Makes all columns not editable for a specific row.
Hide a column	(ListviewHideColumn, "xxxxxxxx", "nnn")	Hides a specific column in a list view and removes the column from the Field Chooser. This is useful if a specific column is to be 'hidden' because of some security setting.
Show a column	(ListviewShowColumn, "xxxxxxxx", "nnn")	Shows a specific column in a list view.
Hides columns	(ListviewHideColumns, "xxxxxxxx", "nnn")	Removes columns from the list view. All columns from the specified column will be hidden. All columns numbered before the specified column will be shown.
Set focus to the listview	(ListviewSetFocus, "xxxxxxxx", " ")	Sets focus to the listview.
Set focus to a row	(ListviewSetFocusRow, "xxxxxxxx", "nnnn")	Sets the specified row to be selected.
Get footer values	(ListviewGetFooterValues, "xxxxxxxx", " ")	Returns the values in footer columns, separated by TAB.
Begin edit at	(ListviewBeginEditAt, "xxxxxxxx", "nnn row")	The returned format is nnnxxxxx, where nnn is the column index and xxxx is the footer text.
Get row count	(ListviewGetRowCount, "xxxxxxxx", " ")	Starts editing at the specified column and row index. The listview must be editable.
Set a dropdown list for a column	(ListviewSetDropDownList, "xxxxxxxx", "nnn list")	Returns the number of rows.
Set the list of constraints for the specified column		This sets the list of constraints for the specified column. The items in the list must be separated by a semi-colon. Optionally, you can define numeric enumerators for each item using the [nn] syntax, for example: England[0];Canada[1];America[2];Australia[3]; If enumerators are not supplied, then the items will be numbered from 1 upwards.
Get the page size	(ListviewGetPageSize, "xxxxxxxx", " ")	Returns the page size selected by the user. This callout should be used if the 'Allow page size' option is switched on and the application needs to know how many rows were selected to be returned to populate the list view. The return value will contain 9999 if 'ALL rows' is selected.
Set a translated dropdown list for a column	(ListviewSetTranslatedList, "xxxxxxxx", "nnn list")	This sets the list of constraints for the specified column; each item will be translated. The items in the list must be separated by a semi-colon. Optionally, you can define numeric enumerators for each item using the [nn] syntax, for example: Stocked[1];Non-stocked[7];Freight charge[4];Miscellaneous charge[5]; If enumerators are not supplied, then the items will be numbered from 1 upwards.
Disable a toolbar button	(ListviewTbarDisable, "xxxxxxxx", "nnnnn")	This disables a button in the listview toolbar.
Enable a toolbar button	(ListviewTbarEnable, "xxxxxxxx", "nnnnn")	This enables a button in the listview toolbar.
Toggle checkmark state for a toolbar button	(ListviewTbarToggle, "xxxxxxxx", "nnnnn")	This toggles the checkmark for a button in the listview toolbar.
Set checkmark a toolbar button	(ListviewTbarSetCheckMark, "xxxxxxxx", "nnnnn")	This sets the checkmark for a button in the listview toolbar.

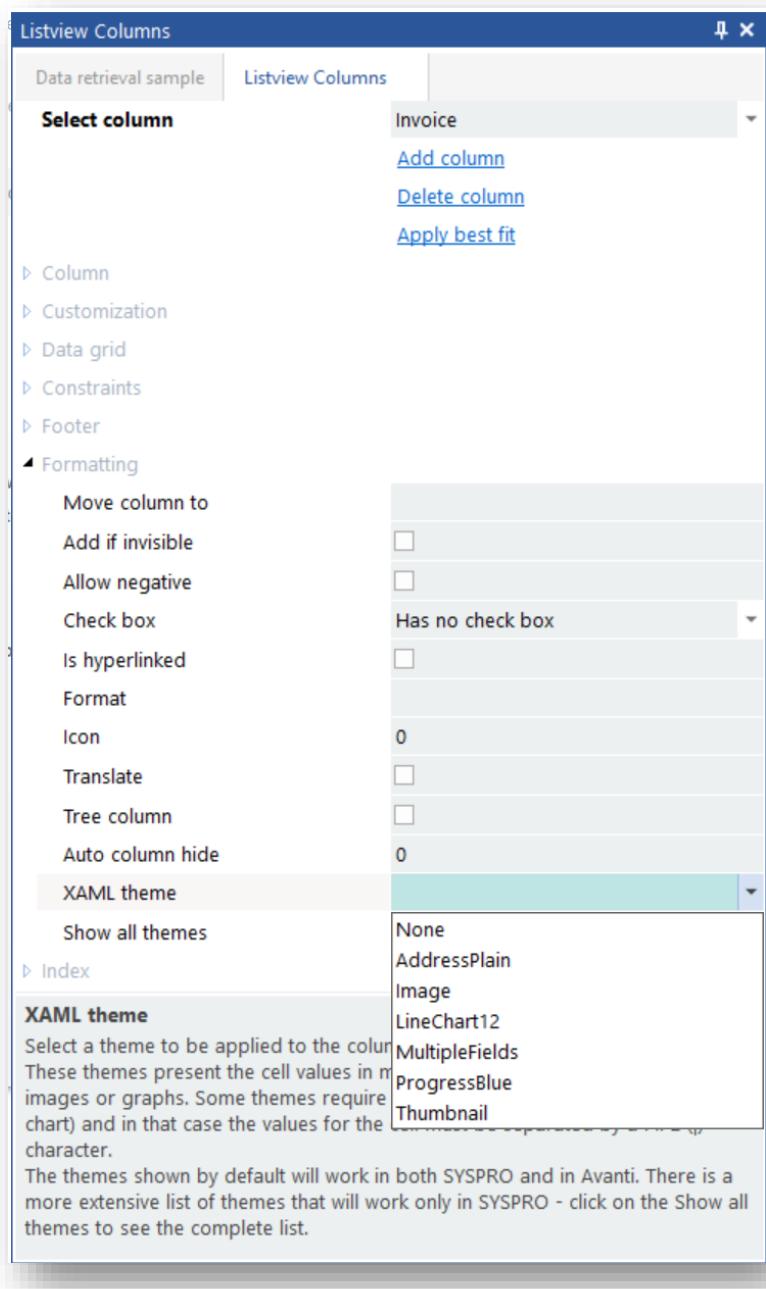
LIST VIEW COLUMN AND XAML

The data within a listview column can be rendered by a standard SYSPRO XAML theme.

This is done by selecting a XAML theme in the formatting section of the column definition.

For example:

- **Image** theme will render an image URL in the cell.
- **ProgressBlue** theme takes a number between 1 and 100 and shows a progress bar.
- **MultipleFields** takes a coma separated list of up to 9 pieces of data and displays each one on a different line.

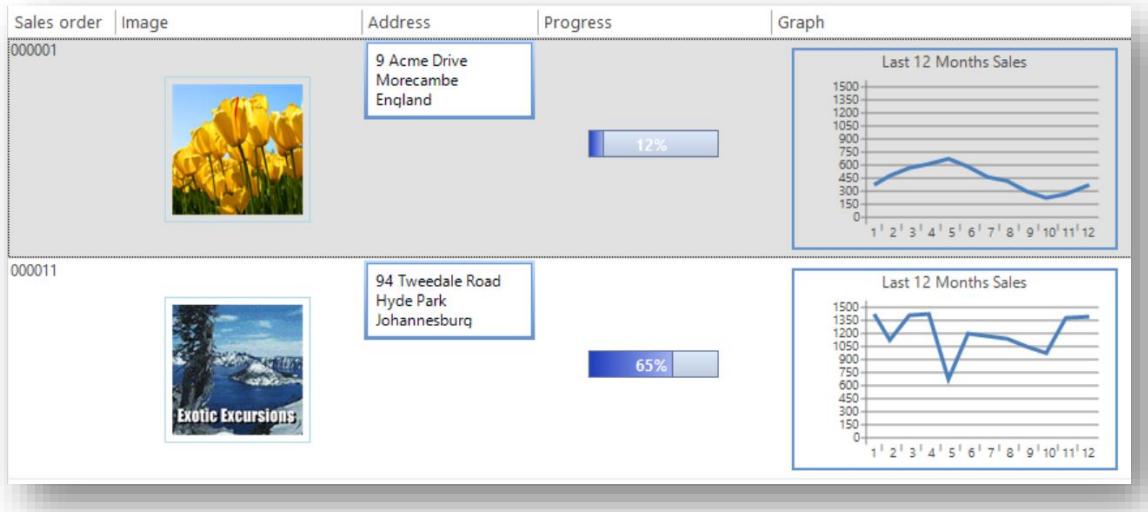


These themes are held in base/Samples and are named Form_?????.XAML (e.g. Form_ProgressBlue.XAML).

Variables should be pipe separated (|) and are referenced in the Theme as %1% %2% etc.

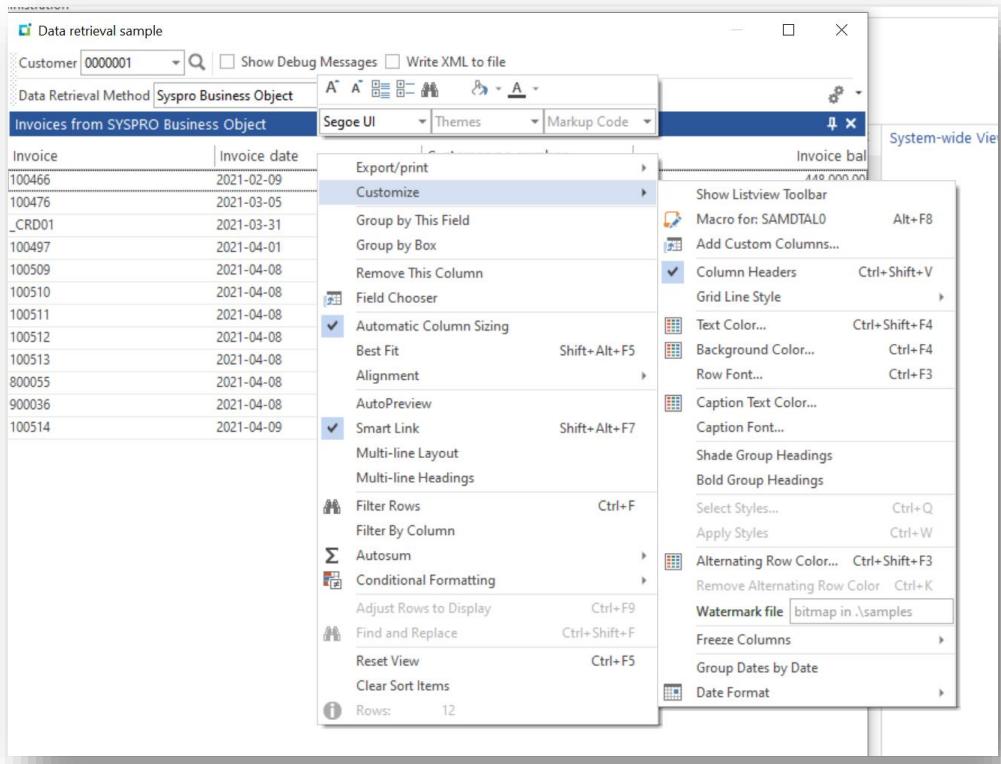
Some of these XAML themes have equivalent HTML themes so that they work in both SYSPRO and in Avanti – these are shown in the drop-down list when the **Show all themes** option is disabled.

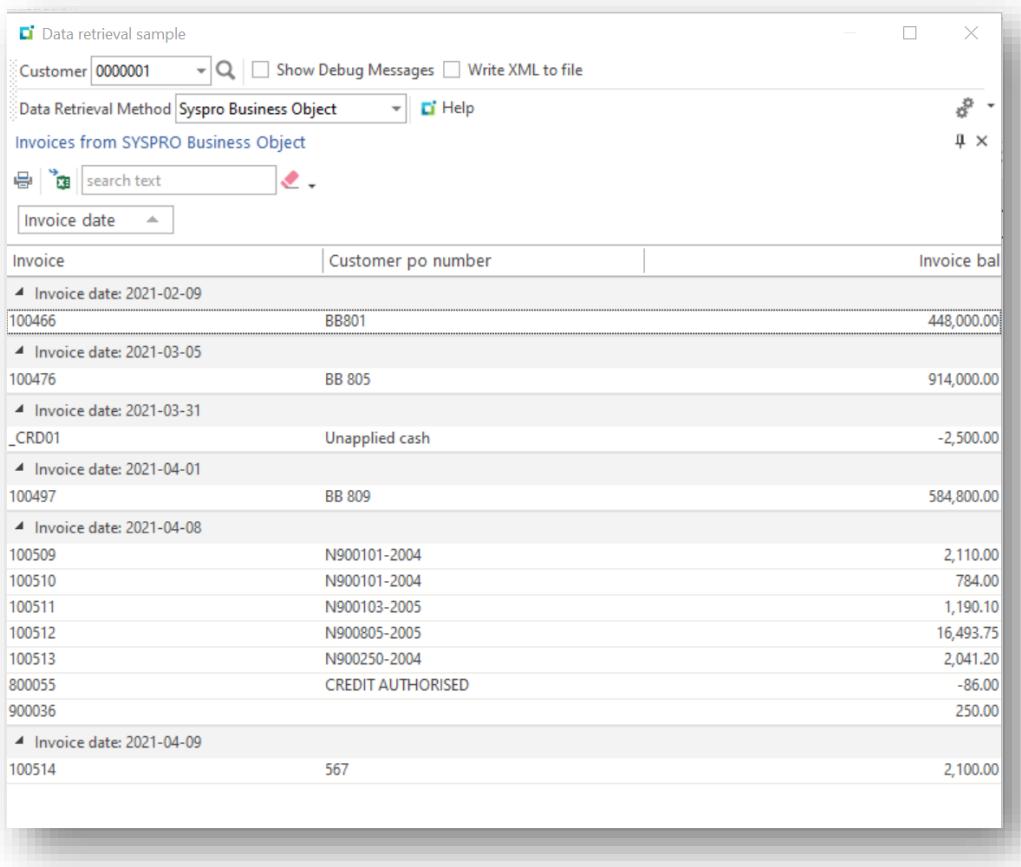
An example of a listview using themes:



LIST VIEW BEHAVIOR

The listview in the Application Designer is the same control that the standard SYSPRO system uses and has all the same behaviors such as sorting, filtering, and formatting.





RETRIEVING LISTVIEW CONTENT

There are two ways of returning the contents of a listview from a script using the callouts

- **ListviewGetRecords**
- **ListviewGetRecordsAsXML**

LISTVIEWGETRECORDS

This callout returns a 2-dimensional array of rows and columns.

Here's example code that shows how to process this array:

```
dim ReturnValue, rowIndex, columnIndex
ReturnValue = CallSYSPROFunction(ListviewGetRecords, "DTS004I0", " ")
For rowIndex = 0 to ubound(ReturnValue) -1
    For columnIndex = 0 to ubound(ReturnValue(rowIndex))
        MsgBox returnValue(rowIndex)(columnIndex)
    Next
Next
```

LISTVIEWGETRECORDSAsXML

This callout returns an XML string.

If the listview contents look like this:

Invoices			
Sales order	Invoice	Invoice date	Customer PO Number
000793	100466	09 Feb 2015	BB801

then this will be the XML output:

```
<Records><Record><SalesOrder>000793</SalesOrder><Invoice>100466</Invoice>
<InvoiceDate>2015/02/09</InvoiceDate>
<CustomerPONumber>BB801</CustomerPONumber></Record></Records>
```

FORM CONTROL

The Form control used by the Designer is the primary User Interface control that the main SYSPRO application uses and as such inherits all the SYSPRO user interface look, feel and behavior. This means that any application using this control will look the same as the core SYSPRO product.

Adding a Form control to the application creates a Form Pane. There can be multiple forms in an application and each form can have its own toolbar.

Refer to the [toolbar section](#) for more detail.

Once a form pane has been created, the form content must be designed. The layout for all the form panes is the same. It is a two-column layout with a form field on each row. Column one for the description/caption and column two for the data.

The **Design Form** function enables the addition of an extensive list of field types to create the form.

ADDING THE FORM CONTROL

Clicking on the Form Control in the Toolbox will add a form pane.

The first decision is to decide if the form is a dialog box. A dialog box is a Form pane that (when opened by VBScript code) must be closed by the user to continue with the application. It is often used to force data entry and to control the user flow through an application.

Next, you should decide if the form is to be editable, that is, is the form being used to display data only (e.g. an enquiry screen) or is it being used to enter and amend data?

Note that the form pane must be **enabled** to become editable. This can be done by VBScript or by the Form property **Enabled on initial load**:

Form Details - Customer Information

Pane Properties

Form ID	APP001FO
Pane ID	001
Toolbar ID	
Form title	Customer Information
Show pane as dialog box	<input type="checkbox"/>
Disable close button	<input type="checkbox"/>
Dialog box size	Small
Pane can be closed	Yes
Pane can be hidden	<input checked="" type="checkbox"/>
Pane can be floated	<input checked="" type="checkbox"/>
Pane can be docked	<input checked="" type="checkbox"/>
Show pane caption	<input checked="" type="checkbox"/>
Custom form pane	<input type="checkbox"/>

[Delete pane](#)
[Design Toolbar](#)
[Design Form](#)
[Import Form](#)

Form Properties

Editable	<input checked="" type="checkbox"/>
Enabled on initial load	<input checked="" type="checkbox"/>
Review-type form	<input type="checkbox"/>
Cannot save form values	<input checked="" type="checkbox"/>
OnAfterSubmit event supported	<input type="checkbox"/>
Disallow field properties	<input type="checkbox"/>
Load previous form values	<input type="checkbox"/>
Press <CR> action	
Post <CR> event once only	<input type="checkbox"/>

Avanti Properties

Disable auto save	<input type="checkbox"/>
-------------------	--------------------------

VBSHRIPT CALLOUTS

There is an extensive list of VBScript callouts to control the form and its fields.

Any field value can be read into the VBScript or set by the VBScript using these callouts:

Form callouts		
Set focus to a field	(FormFieldSetFocus, "xxxxxxxx", "caption")	Leave the caption blank to set focus to the first editable field on the form.
Update a form field value or its properties	(FormFieldSetValue, "xxxxxxxx", "caption string")	Requires the name of the form, the caption of the form field and a string (separated by). The string can be a simple value, or it can be an XML string that changes the field's properties: <Field Value='abc' CaptionBackground='blue' CaptionForeground='white' Height='100' CaptionIsBold='true' IsBold='true' IsItalic='true' Icon='nnn' ></Field>
Get a form field value	(FormFieldGetValue, "xxxxxxxx", "caption")	Returns the value of the form field.
Disable a form field	(FormFieldDisable, "xxxxxxxx", "caption")	
Enable a form field	(FormFieldEnable, "xxxxxxxx", "caption")	
Enable the form	(FormStartEditing, "xxxxxxxx", " ")	Enables the form and starts editing at the first editable field on the form.
Disable the form	(FormStopEditing, "xxxxxxxx", " ")	Disables the form.
Change a caption	(FormFieldChangeCaption, "xxxxxxxx", "caption new caption")	Requires the name of the form, the caption of the form field and the new caption (separated by).
Hide a form field	(FormFieldHide, "xxxxxxxx", "caption")	
Show a form field	(FormFieldShow, "xxxxxxxx", "caption")	
Clear the form of all	(FormClearData, "xxxxxxxx", " ")	
Set a form field tooltip	(FormFieldSetTooltip, "xxxxxxxx", "caption new tooltip")	Requires the name of the form, the caption of the form field and the new tooltip (separated by).
Update dropdown list	(FormUpdateDropDown, "xxxxxxxx", "caption list of items")	Requires the name of the form, the caption of the form field and a list of items (separated by). The items must be separated by the ; character.

Most of the callouts are self-explanatory but the callout **FormUpdateFromXML** may require further explaining.

FORMUPDATEFROMXML CALLOUT

This callout can populate a form with values derived from an XML string.

This allows you to populate an entire form with a single callout instead of having to use the **FormFieldSetValue** callout for multiple fields.

Update all fields on a form	(FormUpdateFromXML, "xxxxxxxx", xmlForm)	The XML string can contain as many form fields for which values and properties are to be updated. The form can also be updated with a new set of fields.
Get a form field v	Select Control Name FORM01F0 Travel Request Form	Returns the value of the form field.

When the Form control name is selected, the script editor will construct the XML string consisting of the names of all the fields on the form and insert it into the script.

You just have to define the values for each form field:

```
dim xmlForm
xmlForm=<Form ClearData='true' ClearForm='false'><Fields>" &_
"<Field Name='Date requested' Value=' ' />" &_
"<Field Name='First name' Value=' ' />" &_
"<Field Name='Last name' Value=' ' />" &_
"<Field Name='Date of birth' Value=' ' />" &_
"<Field Name='Destination' Value=' ' />" &_
"<Field Name='Departure date' Value=' ' />" &_
"<Field Name='Travel purpose' Value=' ' />" &_
"<Field Name='Currency' Value=' ' />" &_
"<Field Name='Airfare' Value=' ' />" &_
"<Field Name='Transportation' Value=' ' />" &_
"<Field Name='Hotel accommodation' Value=' ' />" &_
"</Fields></Form>"
returnValue = CallSYSPROFunction(FormUpdateFromXML, "FORM01F0", xmlForm)
```

The XML element **ClearData** indicates if the form is to be cleared of all data before any fields are updated with values. The default is ‘true’, but if set as ‘false’ then the form is not cleared of any data first.

The XML element **ClearForm** indicates if the form is to be completely cleared of all form fields. Using a value of ‘true’ allows you to dynamically build a form on-the-fly.

returnValue will be blank if the incoming XML is well-formed.

If the XML is not well-formed then **returnValue** will contain an error message.

FORM EVENTS

The form can raise the following events:

Fires when a form field value changes.	FormValueChanged	Param1 contains the form field caption (in lower-case), Param2 contains the changed value and Param3 contains the type of field changed (A=Alpha, B=Check box, D=Date, E=Color, F=Font, I=Picture, 1=Multiline, 2=Spin, 3=Slider, 4=RadioButton, 5=Hyperlink, N=Numeric, L=DropDown, K=DropDownEntry). Example: if EventType = FormEvent and EventID = FormValueChanged then Param1 contains the form field caption (in lower-case).
Fires when clicking on a form field hyperlink.	FormHyperLink	
Fires when clicking on a form field button.	FormButtonClicked	Param1 contains the form field caption (in lower-case) and Param2 contains the current field value.
Fires when a form field has focus.	FormGainFocus	Param1 contains the field caption. Note that the OnGainFocus event must be checked against the Application Details for this event to fire.
Fires when ENTER pressed.	FormENTERPressed	

FIELD TYPES

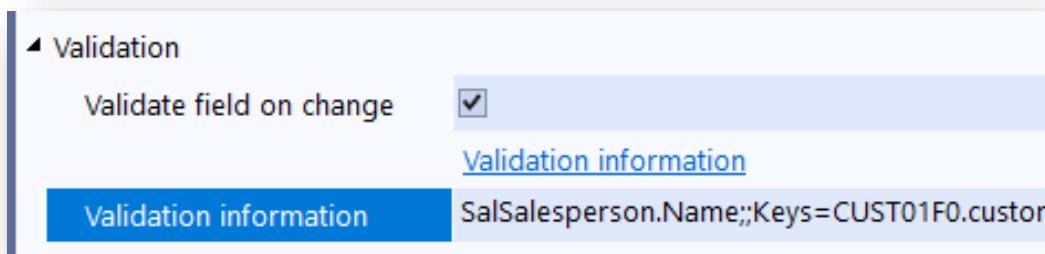
When the **FormValueChanged** event is raised, **Param3** holds the field type.

- A = Alpha
- B = Check box
- E = Color Picker
- D = Date
- L = Drop down
- L = Drop down Entry
- F = Font picker
- 5 = Hyperlink
- 1 = Multiline
- N = Numeric
- 4 = Radio Button
- 3 = Slider
- 2 = Spin Button
- 6 = Time
- 7 = Time span

The **Picture viewing** and **Group Heading collapse and expand** actions do not raise events.

VALIDATE FIELD ON CHANGE

You can validate a key form field without using any script logic. In the form field properties for a form field, you will see the property **Validate field on change**:



This facility allows you to specify the name of the table for which the form field is to be validated against.

Click on Validation information to select a table, a column to be returned as the description, and the error message to be displayed if the KEY is not valid.

This will refresh the **Validation information** property, as shown below (as an example):

```
SalSalesperson.Name;;Keys=CUST01F0.customer  
branch,CUST01F0.salesperson;;Msg=Branch '%key1%', Salesperson '%key2%' not on file;;
```

What's noticeable in this syntax is that you can supply up to 5 keys to derive the keys from. In the case of the **Salesperson** field, the validation for the key field should be the Branch and then Salesperson, as in this syntax: `Keys=CUST01F0.customer branch,CUST01F0.salesperson`. Also, you'll notice that the error message can contain placeholders for each of those key fields (%key1%, %key2% ...).

The syntax `SalSalesperson.Name` defines the table to validate and the column name to be returned as the description.

If the key is valid, the form field is updated with the description returned.

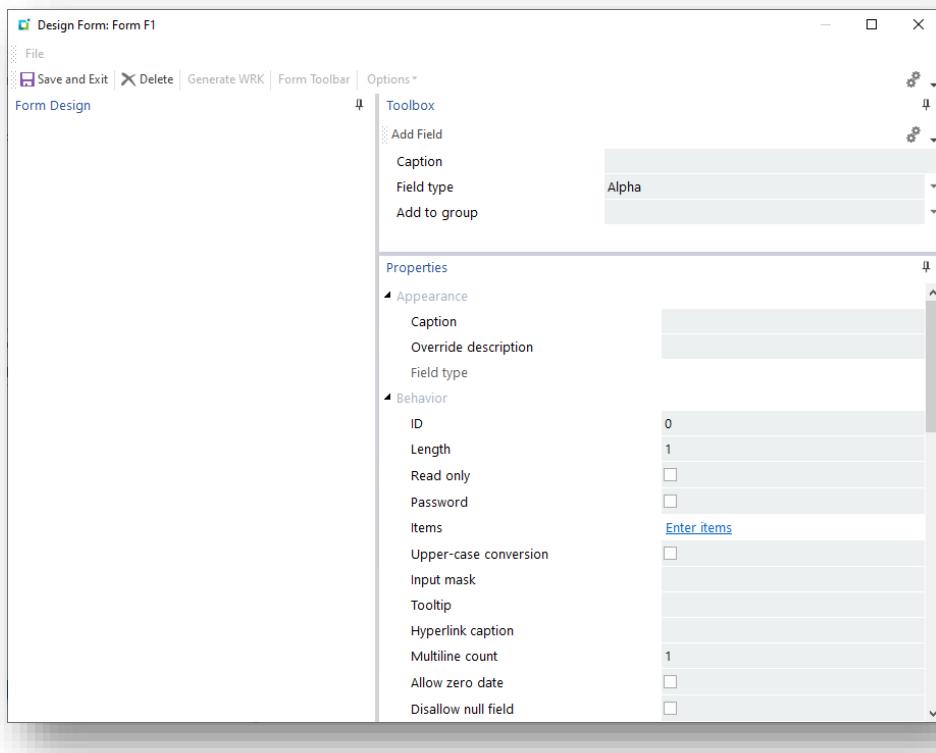
If not valid the error message is displayed, the tooltip against the field is cleared out and focus is set back on the field.

This is how the Salesperson field would look like with the Salesperson's name:

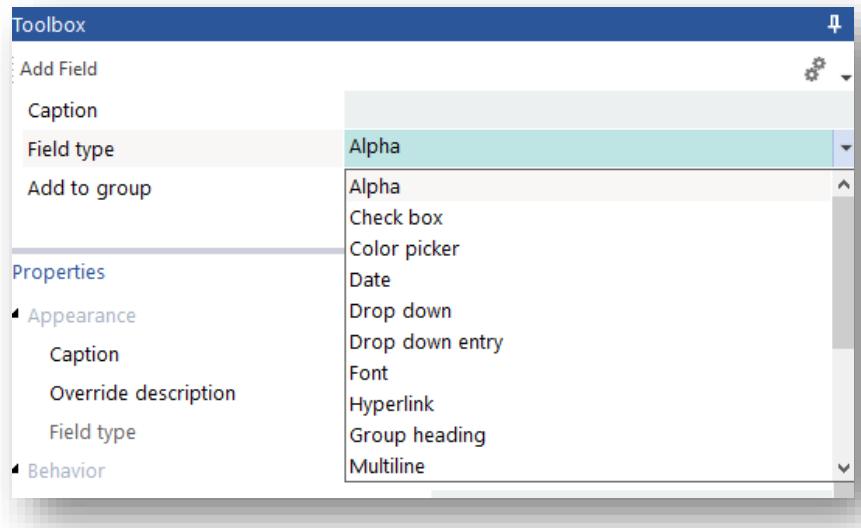
The screenshot shows a software interface titled "Customer Maintenance using Data Source". The main window is labeled "Customer Details". At the top, there is a toolbar with icons for file operations (New, Open, Save, Close) and navigation (Search, Back, Forward). A dropdown menu shows "Customer 0000001". On the right side of the toolbar is a "Help" button and a gear icon for settings. Below the toolbar, the "Customer Details" section has a header row with columns for "Customer" (containing "0000001") and "Name" (containing "Bayside Bikes"). The "Name" field is highlighted with a blue selection bar. Subsequent rows show other customer details: "Branch" (Receivables - North), "Currency" (\$), "Salesperson" (Sandra Jenkins), "Invoice terms" (30 Days - 2.5%), and "Geographic area" (Northern Region). Each row has a search icon on its right. At the bottom of the window, a status bar displays "Editing customer: Bayside Bikes".

FORM DESIGN

Once the initial Form pane has been created the next step is to create the form content. Clicking on **Design Form** will launch the form design screen.



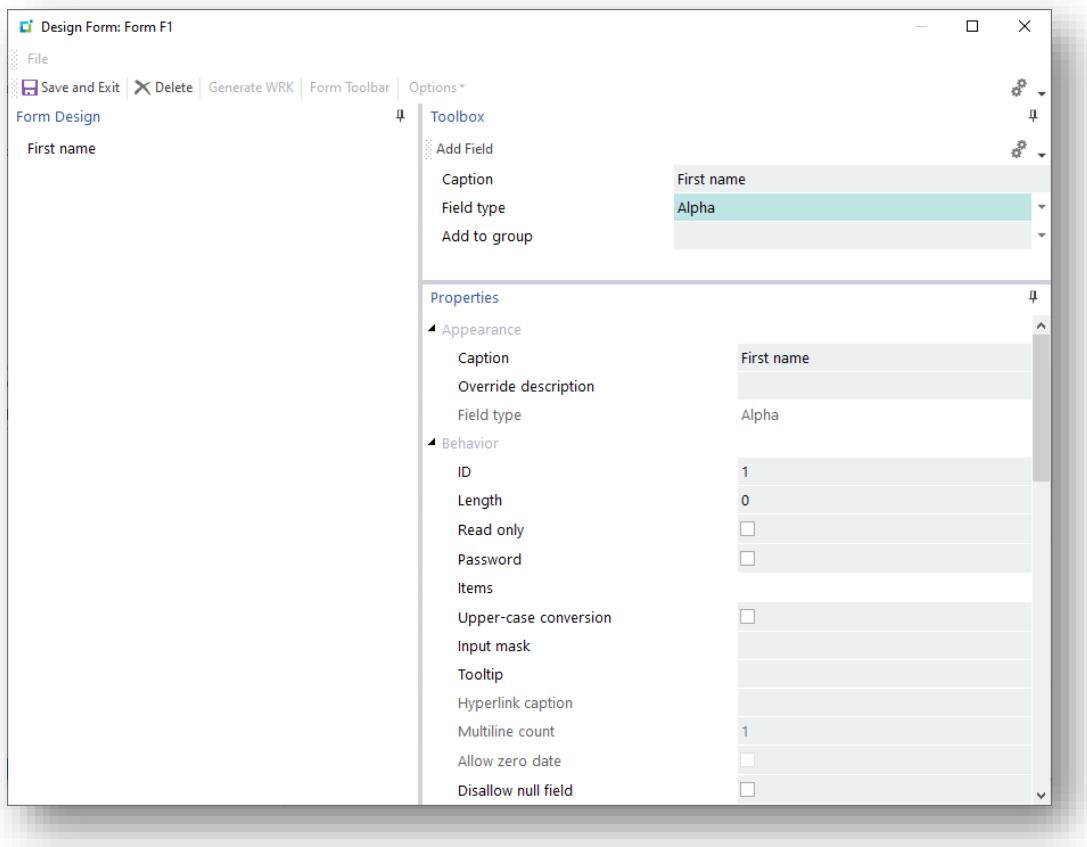
There is an extensive list of Field Types that can be added to a form and each Field Type has its own set of properties that can be set.



These are designed to alter its look and behavior.

ADDING FIELDS

Enter a Caption, select a field type, decide if it needs to be added to an existing Group Header Control and then press the **Add Field** button on the toolbar. The field will appear in the Form Design window, it will be assigned a unique ID and at the same time the field properties window will be populated with the field details and any field specific properties enabled.



CAPTION CONSIDERATIONS

Please note that under normal circumstances, the Caption should be unique to a form.

If in the event it is necessary to have two fields on the form with the same caption, then you need to use the **Override Description** property to create a unique reference. The override description is then used in any VBScript. It should also be used if the caption is to be left blank.

See [Form Design](#) on SYSPRO's online Help for more information.

If the caption matches the caption of a SYSPRO key field, then the field inherits the behavior of that key field. For example, setting an Alpha field caption as **Customer** will automatically add a browse button for a customer browse (and Predictive Search will be enabled) and when the application runs a right-click will display all the options available in SYSPRO when right-clicking a customer.

You can click on the **Event IDs for Keys** hyperlink in the Application Details to view all the available Captions for **key field**. This list will indicate what captions you need to use.

For example:

For a BIN LOCATION you'll need the word **Bin**.

Event IDs for Keys		
Caption	Toolbar event ID	Search tablename
Asset expense	38051	AssetExpenseCode
Asset group	38049	AssetGroup
Asset location	38052	AssetLocation
Asset owner	38055	AssetOwner
Asset reason	38056	AssetReasonDisposal
Asset reval reason	38057	AssetReasonRevaluation
Asset status	38125	AssetStatus
Asset type	38053	AssetType
Assetcapexitem		AssetCapexItem
Bank	38026	Bank
Batch number	38153	EftBatch
Bin	38067	InvBin
Blanket p/o contract	38156	BlanketPurchaseOrder
Bomstandardnarration		BomStandardNarration
Botcard		BotCard
RntSearch		RntSearch

Pressing **Save and Exit** on the Form design window returns you to the Application Designer main layout and displays the layout in the form pane.

FIELD DEFAULTS

To set a field default, enter a value into the field whilst in design mode.

Alternatively, use VBScript and the [AppDesigner_OnInitialization\(\)](#) function to set values on the form as it starts up.

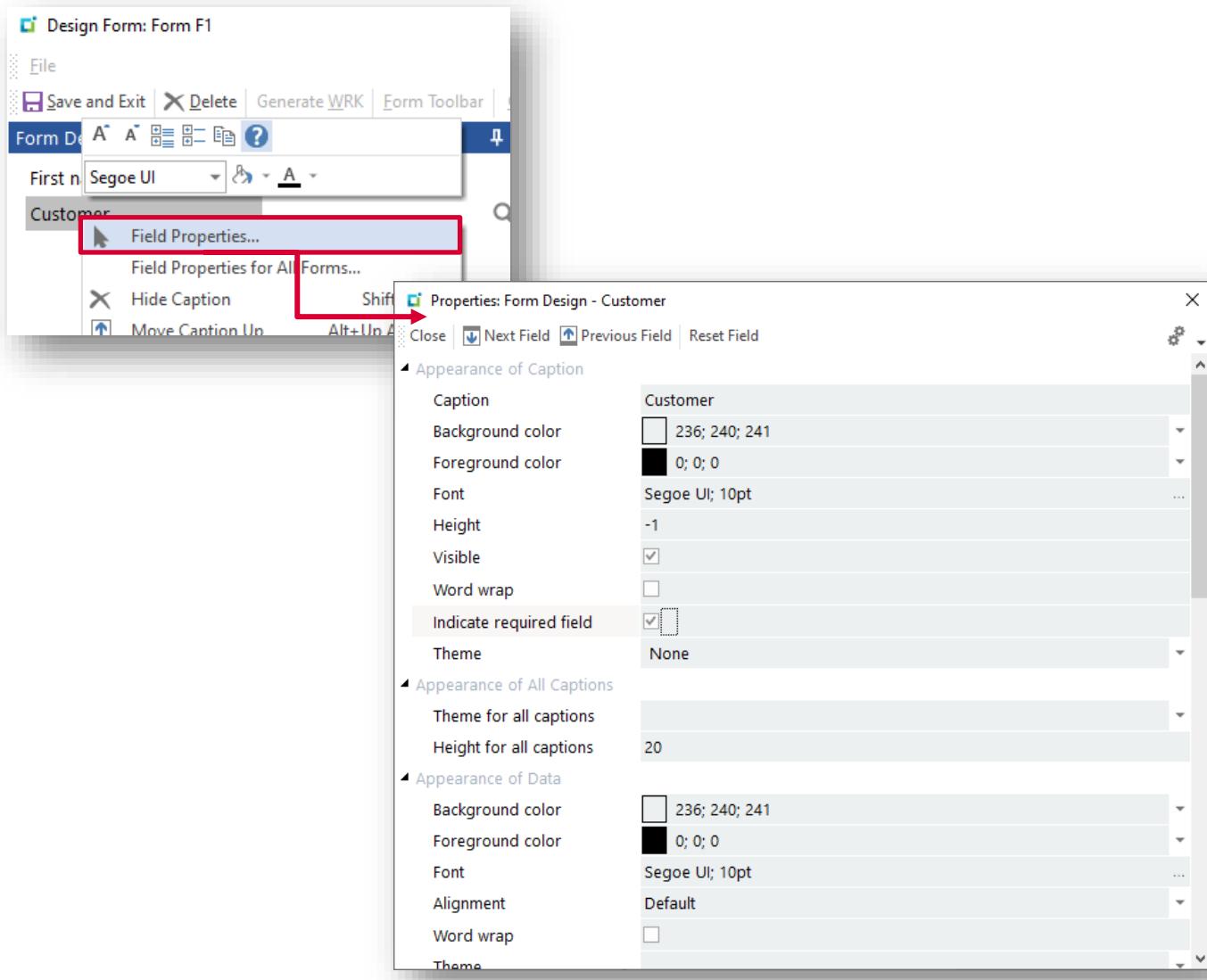
RE-ORDERING FIELDS

The order of the fields on the form can be changed by simply clicking on the required field with the mouse and dragging it up or down the form.

Alternatively, you can press **Alt+UpArrow** or **Alt+DownArrow** to move items up or down. This key also moves items into a Group category.

SETTING FIELD FONTS AND COLORS

For more sophisticated formatting, right-click on any form field and select **Field Properties**:

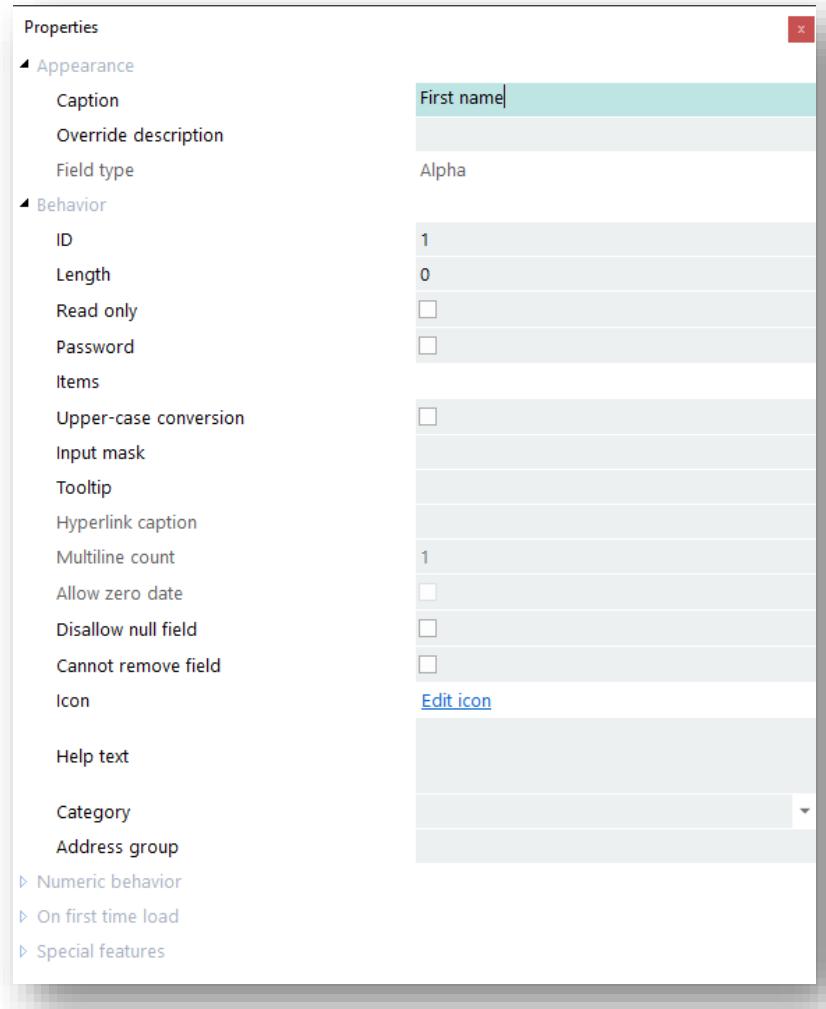


FIELD TYPES

This section describes each of the field types that can be added to a Form Pane.

ALPHA

The Alpha field type is the same as a traditional text box. Available properties include:



The Category property qualifies the type of Alpha field.



For example, setting the category to **FileBrowse (client)** will display a browse button next to the field which (when pressed) will display a file browser browsing the local machine.

CHECKBOX

The checkbox is either checked or not. A value of **1** is checked and **0** is unchecked.

There is a Special features property that allows this to change to Y and N. This is particularly useful when using SYSPRO business objects to populate a form because they use Y and N for checkbox type questions.

If you need to set a default value for the checkbox, then simply check it or uncheck it in the form design.

COLOR PICKER

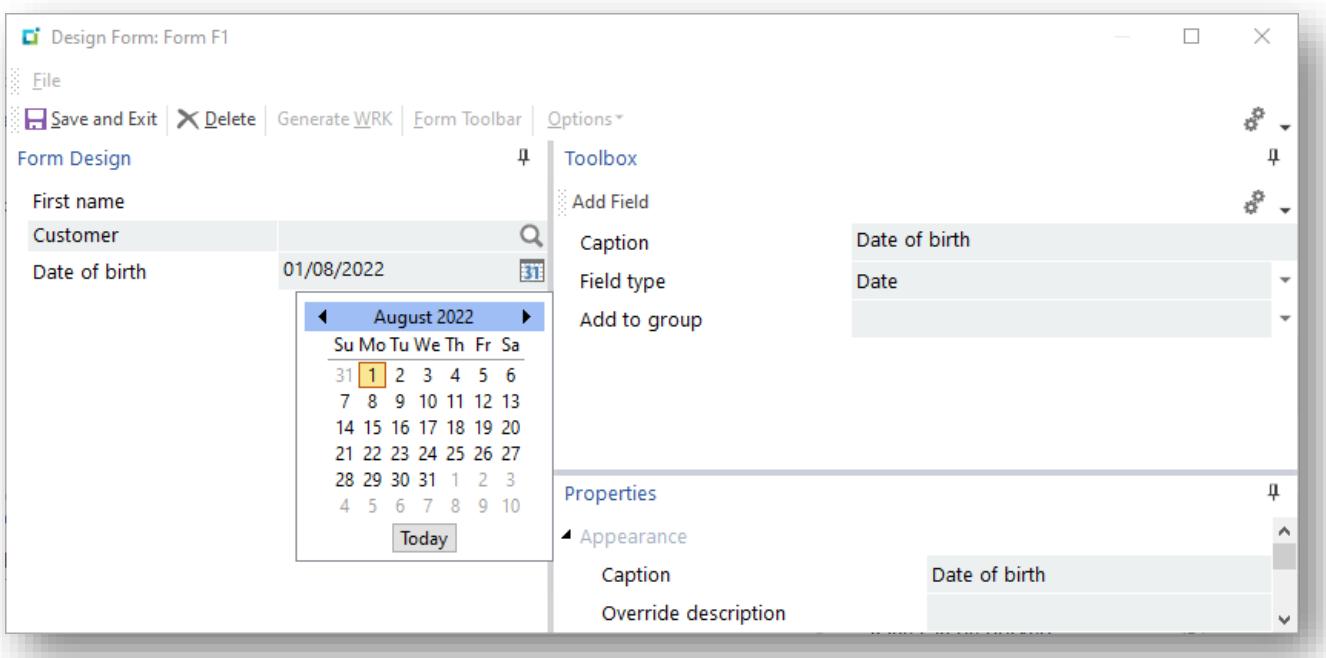
This control allows the selection of an RGB color sequence.

A useful color conversion site is: <https://www.rapidtables.com/convert/color/rgb-to-hex.html>

DATE

A date control that allows the selection of a date from a date picker.

If setting the value of this field via VBScript, then the format must be CCYYMMDD (e.g. 20220620).



The tooltip for this control holds a full description of the date.

See [VBScript hints](#) for creating a date in this format.

DROP DOWN

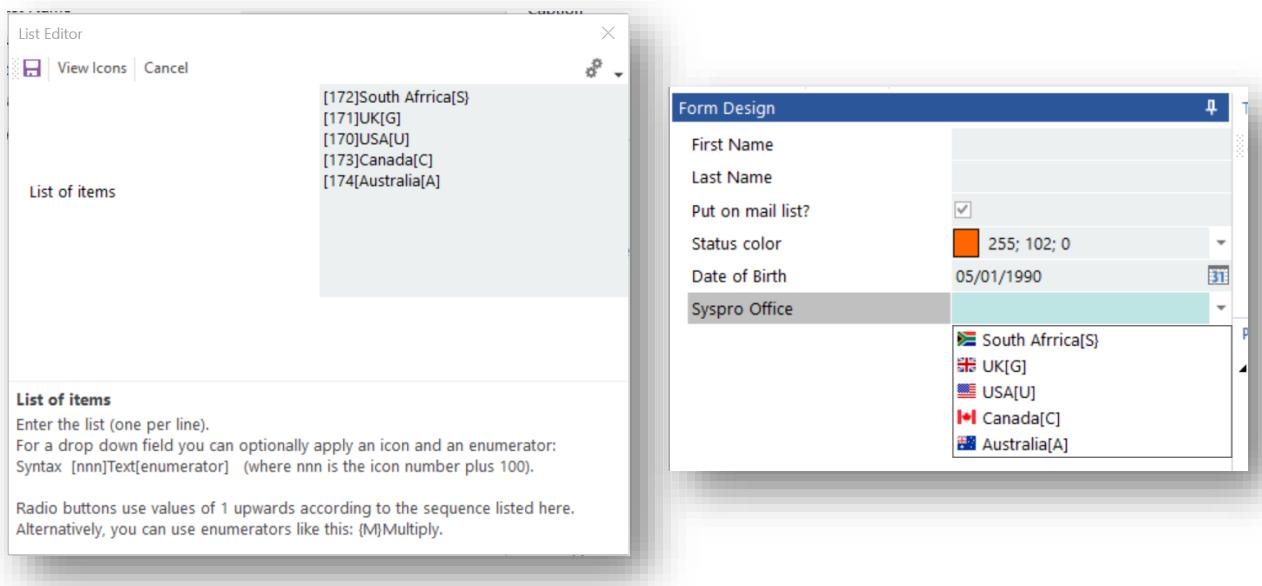
The drop-down field type enables a fixed list of options to be presented.

To enter items in the list, select the **Enter Items** link in the **Behavior** section of the properties. A dialog pane will appear enabling the entry of items in the list.

The list can have an icon, text and an enumerator specified.

- The icon and text will be displayed in the drop down.
- The enumerator can be used to override the index of the drop down.
If used, then it will return the enumerator to the VBScript rather than the index number of the item in the list.

In design mode, the enumerator is displayed. However, when the form runs in the application, the enumerator is not displayed.

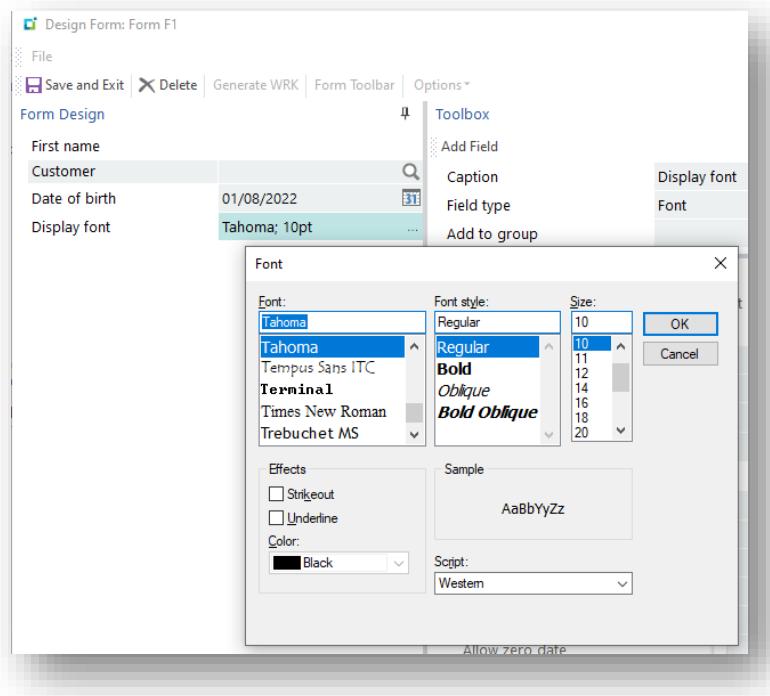


DROP DOWN ENTRY

The drop-down entry field behaves the same as the drop-down field type, except that it allows the user to enter data that is not in the drop-down list.

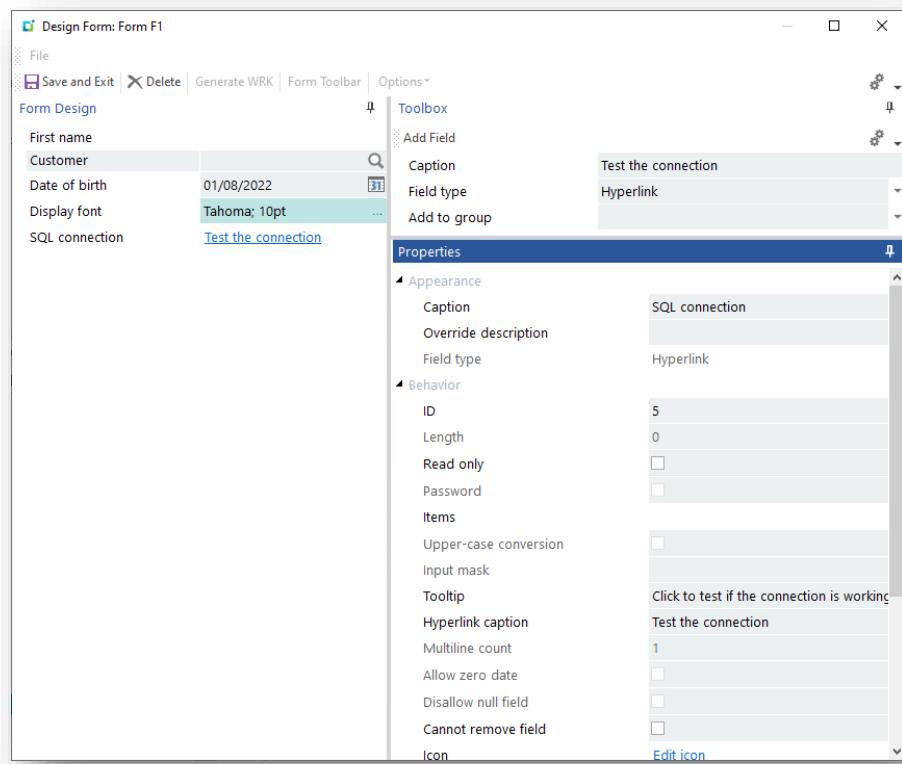
FONT

The Font field type shows a Font picker field. This could be used in VBScript to change the look and feel of different parts of the form:



HYPERLINK

Displays the caption as a hyperlink.



GROUP HEADING

This field allows the grouping of fields into a collapsible section. Once created, it will appear in the **Add to Group** drop down field when creating other fields.

Fields can be moved into a group heading when in design mode by dragging a field to the group. You can also press **Alt+UpArrow** to nudge a field into a group that is above the field, or **Alt+DownArrow** to move into a group that is below. The group can be collapsed automatically by setting the **Show Group Collapsed** property.

MULTILINE

The multiline field allows the display and entry of a multiline text field. The number of lines can be set in the **Multiline count** property.

NUMERIC

This field type is a numeric field. The properties enable the setting of the overall length, the number of decimals and whether to allow commas when editing.

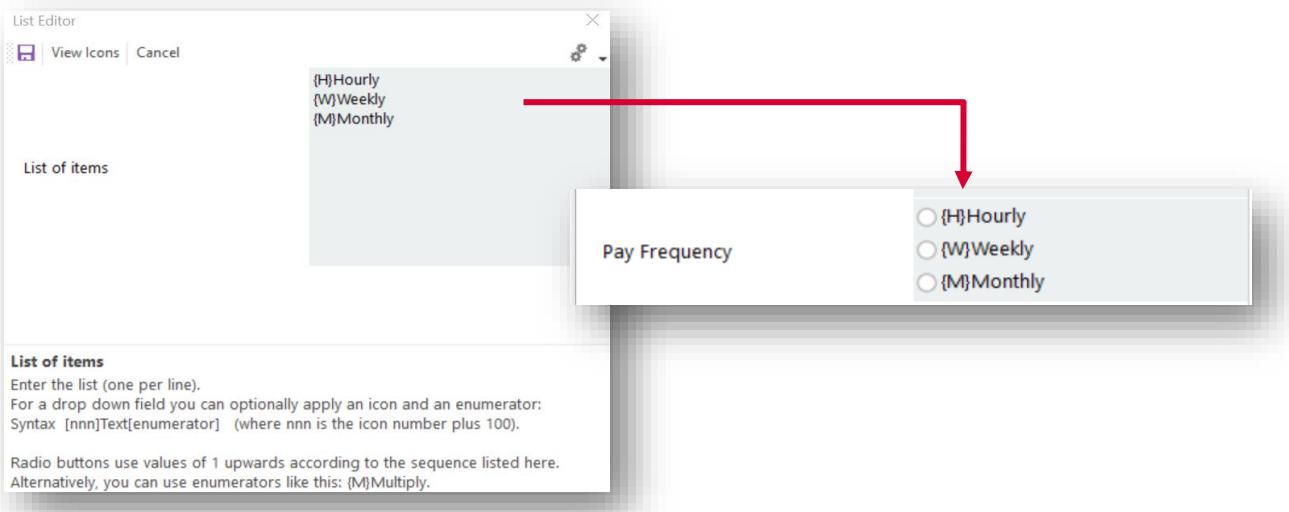
PICTURE

This field type displays a picture. The text in the field should be a URL to a picture. When the form runs, the picture is displayed and (if selected) will be shown in the default picture viewer.

RADIO BUTTON

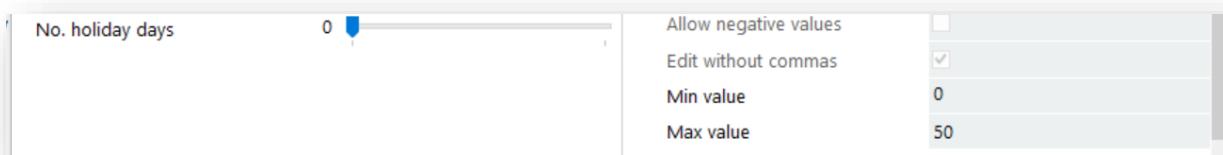
This field type displays a list of items as a series of radio buttons allowing the selection of a single option. The radio button list is created by selecting the **Enter items** option. By default, the value returned to the VBScript is an index number of the item in the list, but an enumerator can be set to override this and return an alpha value to the script.

The enumerator is displayed in the designer but not when the application runs:



SLIDER

This field type is a numeric field that can be set by sliding a pointer along a slider. Maximum and Minimum values can be set.



A screenshot of a slider field configuration. The field is labeled "No. holiday days" with a value of 0. To the right of the slider are four settings: "Allow negative values" (unchecked), "Edit without commas" (checked), "Min value" (0), and "Max value" (50).

SPIN BUTTON

This field type is a numeric field that can be increased and decreased by arrows of a spin control. Maximum and Minimum values can be set.



A screenshot of a spin button field configuration. The field is labeled "Review Month" with a value of 0. To the right are two settings: "Min value" (1) and "Max value" (12). There are also up and down arrow buttons to adjust the value.

TIME

A 24-hour time format field allowing entry and display of hh:mm data.

Hours and minutes are validated automatically.

TIME SPAN

Display days, hours, minutes format.

SETTING FIELD VALUES

When setting field values, the caption is separated from the value by a pipe ("|") sign.

If this is omitted, then the field will not be populated.

For example:

```
call CallSYSPROFunction(FormFieldSetValue, "FRMTSTF0", "Name|" & strName)
```

SETTING A DATE VALUE

A Date field must be in the **CCYYMMDD** format.

- Both the Month and Day must be two characters
- Single digit numbers must have a leading zero

An example of building a date in this format is:

```
dim ccyy: ccyy = year(date)
dim mm : mm = right(string(2,"0") & month(date),2)
dim dd : dd = right(string(2,"0") & day(date),2)
dim ccyyymmdd: ccyyymmdd = ccyy & mm & dd
```

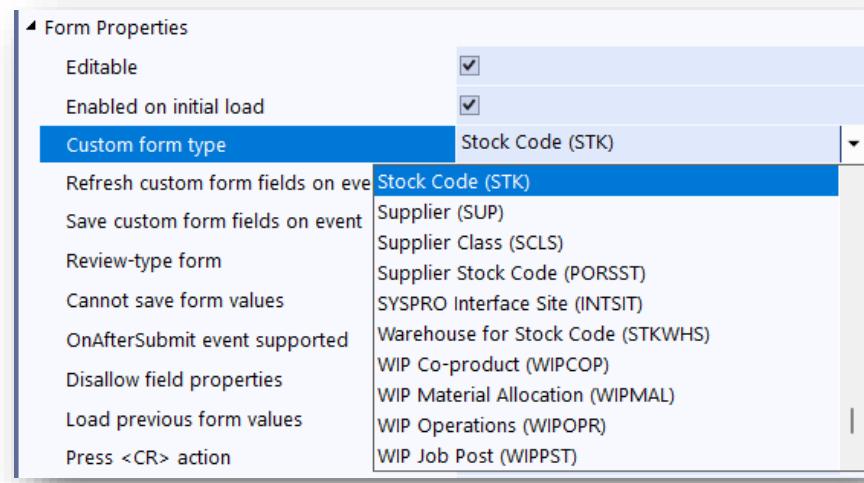
```
call CallSYSPROFunction(FormFieldSetValue, "ANDIEXF2", "Transaction
date|" & ccyyymmdd)
```

HOW TO ENABLE CUSTOM FORM FIELDS ON FORMS FOR END-USERS

Many programs in SYSPRO allow the user to add a mixture of Custom Form fields and Table Master fields to forms. Typically, this makes sense when the majority of fields on a form are associated with master table information.

If you wish your application to enable this facility then you will need to do the following:

- Select a suitable custom form type to be associated with the form:



- Define when to refresh the custom form values.

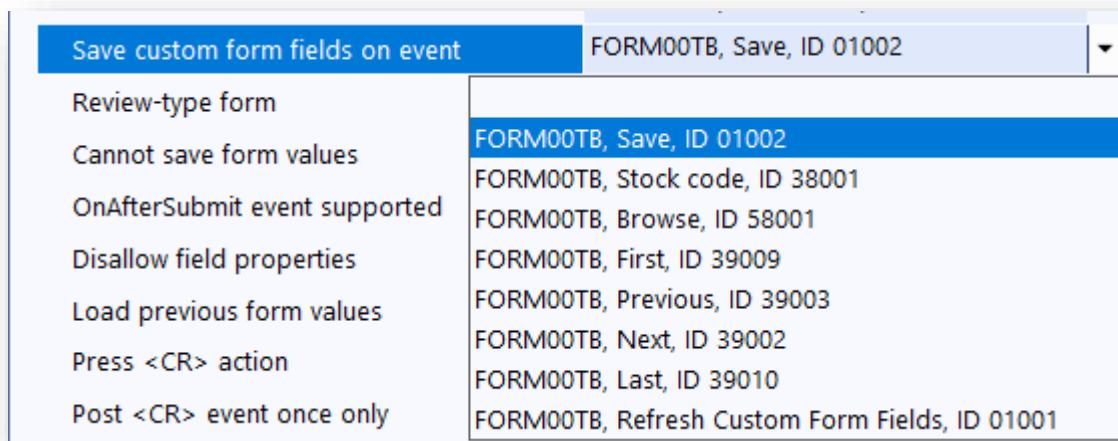
Custom form values can be refreshed either by linking a form's fields to a toolbar event:

Custom form type	Stock Code (STK)
Refresh custom form fields on event	FORM00TB, Stock code, ID 38001
Save custom form fields on event	
Review-type form	FORM00TB, Stock code, ID 38001

or by executing the script callout **FormRefreshCustomFields**:

Refresh custom form fields	(FormRefreshCustomFields, "xxxxxxxx", "key")	Refreshes the values of custom or table fields that have been added to the form by the user. This will only work if a custom form type has been associated with the form and if a valid key is supplied.
----------------------------	---	--

- Optionally, cater for saving custom form values if the form is editable. Custom form values will be automatically saved by linking a form to a toolbar event. When the toolbar button is executed any custom form field value that has changed will be saved.

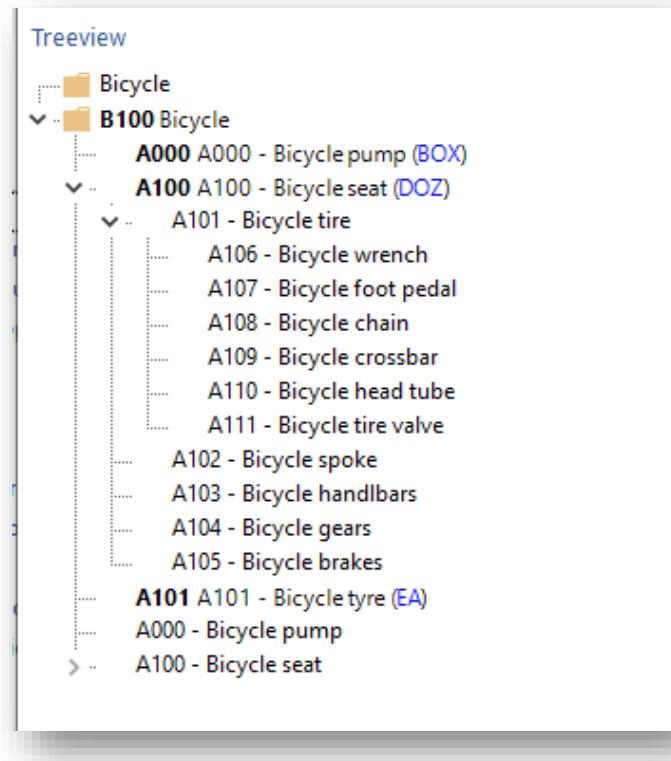


Note: You are allowed only one custom form type to be defined per form.

TREE VIEW CONTROL

The Tree View control is used to display hierarchical representations of items, similar to the way that the files and folders are displayed in the left pane of the Windows Explorer.

Each node may contain one or more child nodes:



These notes explain some of the basics behind the Tree view control.

HOW IT WORKS

The following explains the basic process of a tree view creation:

1. A tree view is built up by adding “Items” to the tree control.
2. Each item has a unique “node index” that the Application Designer automatically allocates to the item when its added.
3. The root node/item of a tree view is always node index **1** and has a Parent attribute of **0** (i.e. it doesn’t have a parent).
4. All other “items” must belong to a parent.

For example:

The following function adds the first level of a tree view:

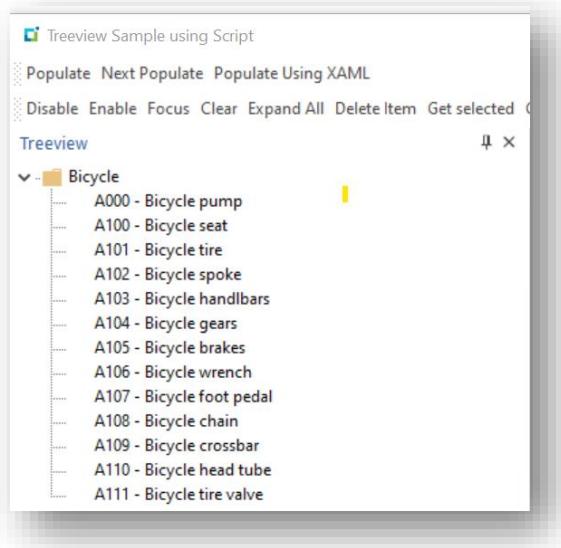
```
function PopulateTreeview()

dim xmlString
xmlString = "<Items>" & _
"<Item Parent='0' Caption='Bicycle' Key='B100' HasChildren='Y' Icon='5' />" &_
"<Item Parent='1' Caption='A000 - Bicycle pump' Value='A000' />" &_
"<Item Parent='1' Caption='A100 - Bicycle seat' Value='A100' />" &_
"<Item Parent='1' Caption='A101 - Bicycle tire' Value='A101' />" &_
"<Item Parent='1' Caption='A102 - Bicycle spoke' Value='A102' />" &_
"<Item Parent='1' Caption='A103 - Bicycle handbars' Value='A103' />" &_
"<Item Parent='1' Caption='A104 - Bicycle gears' Value='A104' />" &_
"<Item Parent='1' Caption='A105 - Bicycle brakes' Value='A105' />" &_
"<Item Parent='1' Caption='A106 - Bicycle wrench' Value='A106' />" &_
"<Item Parent='1' Caption='A107 - Bicycle foot pedal' Value='A107' />" &_
"<Item Parent='1' Caption='A108 - Bicycle chain' Value='A108' />" &_
"<Item Parent='1' Caption='A109 - Bicycle crossbar' Value='A109' />" &_
"<Item Parent='1' Caption='A110 - Bicycle head tube' Value='A110' />" &_
"<Item Parent='1' Caption='A111 - Bicycle tire valve' Value='A110' />" &_
"</Items>"
call CallSYSPROFunction(TreeAddItems, "TREEVWX0", xmlString)
end function
```

This code creates a root item of **Bicycle**.

It has a node index of **1** and no parent. All the other items are its children.

Their **Parent** attribute is set to **1** (which is the node index of the root node):



This can be represented like this:

Caption	Parent ID	Node Index	Children
Bicycle	0	1	1
A000 - Bicycle pump	1	2	0
A100 - Bicycle seat	1	3	0
A101 - Bicycle tire	1	4	0
A102 - Bicycle spoke	1	5	0
A103 - Bicycle handlebars	1	6	0
A104 - Bicycle gears	1	7	0
A105 - Bicycle brakes	1	8	0
A106 - Bicycle wrench	1	9	0
A107 - Bicycle foot pedal	1	10	0
A108 - Bicycle chain	1	11	0
A109 - Bicycle crossbar	1	12	0
A110 - Bicycle head tube	1	13	0
A111 - Bicycle tire valve	1	14	0

The following indicates how to add children to a node like this:

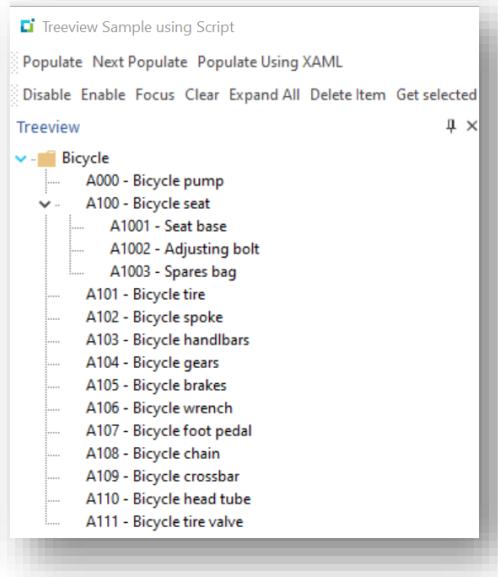
```
function PopulateTreeview2()

dim xmlString
xmlString = "<Items>" &
"<Item Parent='3' Caption='A1001 - Seat base' Value='A1001' />" &_
"<Item Parent='3' Caption='A1002 - Adjusting bolt' Value='A1002' />" &_
"<Item Parent='3' Caption='A1003 - Spares bag' Value='A1003' />" &_
"</Items>"
call CallSYSPROFunction(TreeAddItems, "TREEVWX0", xmlString)
end function
```

This code adds 3 child items to node index 3, which is the bicycle seat.

As it does so, it sets the **HasChildren** attribute of the new parent to **Y**.

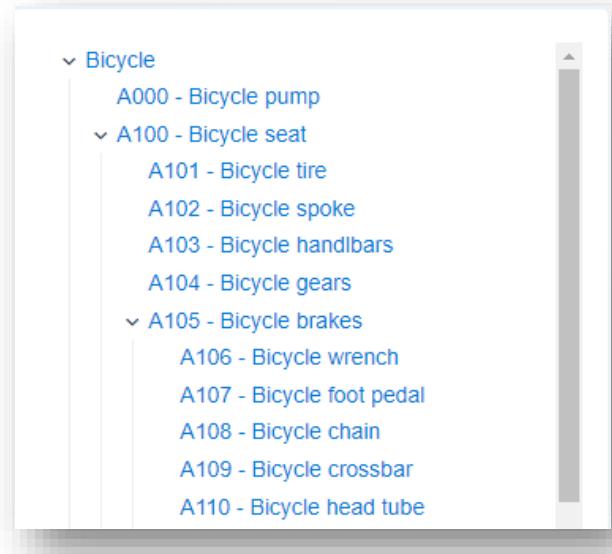
Each new item is allocated a unique node index:



This can be represented like this:

Caption	Parent ID	Node Index	Children
A100 - Bicycle seat	1	3	1
A1001 - Seat base	3	15	0
A1002 - Adjusting bolt	3	16	0
A1003 - Spares bag	3	17	0

Note: If you wish the tree view to appear the same in Avanti, then you should check the tree view option **Multi-level tree view**:



APPLICATION DESIGNER TREE VIEW EVENTS

Every Tree view event returns the following information to the VBScript **OnUserEvent**:

```
Function AppDesigner_OnUserEvent(EventType, EventName, EventID, Param1, Param2, Param3)
```

See **Event Types and Events & Event Ids** on the **Callout Functions** tab in VBScript.

These parameters are explained as follows:

Parameter	Explanation
EventType	This is the "TreeviewEvent" constant, identifying what category of event it is.
EventName	This is the TreeviewID. This identifies which Tree view the events are for. See its Pane Properties
EventID	This will be the event that has just fired (for example: TreeExpand, TreeClick, TreeRightClick, etc.).
Param1	This returns whatever string is in the "Value" attribute
Param2	This returns the node index of the item that invoked the event. E.g. the item that was clicked. This would be used as the parent attribute if adding children.
Param3	This returns: <ul style="list-style-type: none">▪ 1 = Has Children▪ 0 = Has no children

Note: The Value attribute can hold any string. If you need to return more than one value from the tree view, then build it into the value attribute when you build the tree view.

An example taken from the SAMCOC sample app:

```
function PopulateTreeview( m_fullurl,m_alphabet, m_listby)
'NB The value attribute on an <Item> is always returned to the VBScript on param1
of any Treeview event.
'In this example I have two fields delimited by a '|' sign. The word "Cat" is
used to identify an item that is a category.
'I use this when deciding on whether to show a context menu or not. I only want a
context menu on a drink and not a category.
...
...
    for each drink in oDrinks
        TreeviewXML = TreeviewXML & "<Item Parent='1' Caption='"
        drink.strCategory  & "' Value=''" & "Cat|" & drink.strCategory & "'"
        HasChildren='Y' />"
    next
```

OTHER POINTS

To all intents and purposes there is no limit to the size of a tree view.

You can build a complete tree view with levels of parents and children for an initial load. However, data volumes for an application may make this not feasible.

Realistically, it is best practice to get the branches of a tree as and when you need them. Typically, this occurs when you click a node to expand it.

XAML AND TREE VIEW ITEMS

XAML can also be injected into a Tree view item.

To help simplify passing a XAML string to a tree view item and having to substitute the XML special characters, if the < and > characters are replaced with { and }, then the Application Designer will do the substitution for you.

Note: It will not substitute any other special characters that may be in the text, such as Ampersand (&).

If provided as a string to an attribute for a control, you need to make sure that any XML reserved characters are replaced with their relevant encoding. Otherwise XAML and XML get mixed up and nothing will work.

There is a code snippet provided in the Application Designer called

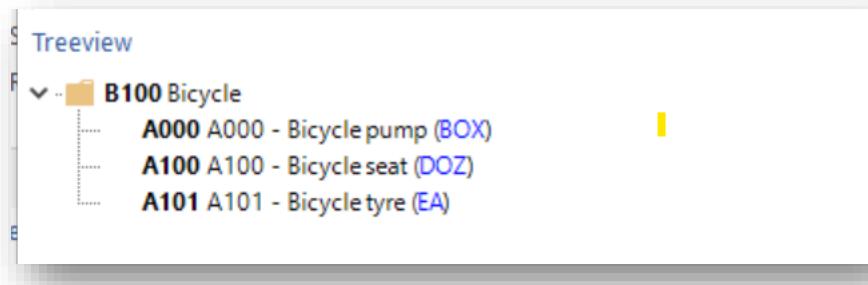
CheckXMLSpecialCharacters which will parse a string and substitute the relevant codes if required.

Therefore, the following code:

```
dim xmlString
xmlString = "<Items>
  & "<Item XAML='&lt;TextBlock&gt;&lt;Bold&gt;B100&lt;/Bold&gt; Bicycle&lt;/TextBlock&gt;' Parent='0' Caption='Bicycle' Value='B100' HasChildren='Y' Icon='5' /> "
  & "<Item Parent='1' Caption='A000 - Bicycle pump' Value='A000' "
  & "XAML='{TextBlock}{Bold}A000{/Bold} A000 - Bicycle pump ({Run Foreground=""Blue"" Text=""BOX""}){/TextBlock}'/> "
  & "<Item Parent='1' Caption='A100 - Bicycle seat' Value='A100' "
  & "XAML='{TextBlock}{Bold}A100{/Bold} A100 - Bicycle seat ({Run Foreground=""Blue"" Text=""DOZ""}){/TextBlock}'/> "
  & "<Item Parent='1' Caption='A101 - Bicycle tyre' Value='A101' "
  & "XAML='{TextBlock}{Bold}A101{/Bold} A101 - Bicycle tyre ({Run Foreground=""Blue"" Text=""EA""}){/TextBlock}'/> "
</Items>"

call CallSYSPROFunction(TreeAddXAMLItems, "TREEVWKO", xmlString)
```

Generates this:



SYNTAX EDITOR CONTROL

The Syntax editor control provides users with a highly sophisticated text editor control that supports many advanced features, including (but not restricted to):

- Text block grouping
- Syntax colorization
- Line numbers
- Font type
- Pre-defined color schemes (SQL, VBScript, XAML, XML)
- Bookmarks
- Break points

The editor is the same one used for the VBScript editor. And using the control is straight forward.

The Callout functions below explain the functionality available.

FEATURES OF THE CONTROL

A mini toolbar (as part of the control) allows the user the following capabilities:

- Cut, Copy and Paste text
- Undo and Redo actions
- Find text
- Zoom text (and print)
- Print text
- Tidy an XML/XAML string
- Enlarge and reduce font size

CALLOUT FUNCTIONS

The Application Designer has the following callout functions for the Syntax Editor:

Syntax editor callouts	
Open a text file	(SyntaxOpenFile, "xxxxxxxx", filename)
Save a text file	(SyntaxSaveFile, "xxxxxxxx", filename)
Clear text	(SyntaxClear, "xxxxxxxx", " ") Clears the text and the Undo buffer.
Insert text	(SyntaxInsertText, "xxxxxxxx", text) Inserts text at the current position.
Get text	(SyntaxGetText, "xxxxxxxx", " ") Returns the text in the editor.
Check for changes	(SyntaxCheckForChanges, "xxxxxxxx", " ") Returns 0 (no changes) or 1 (text has changed).
Set focus to syntax editor	(SyntaxSetFocus, "xxxxxxxx", " ") Focus is set to row 1, column 1.
Set theme	(SyntaxSetTheme, "xxxxxxxx", theme) Sets the theme for the syntax editor. Theme values may be sql, vbscript, xaml or xml. Note that setting a theme will clear out any text.

USING THE CONTROL

The following code:

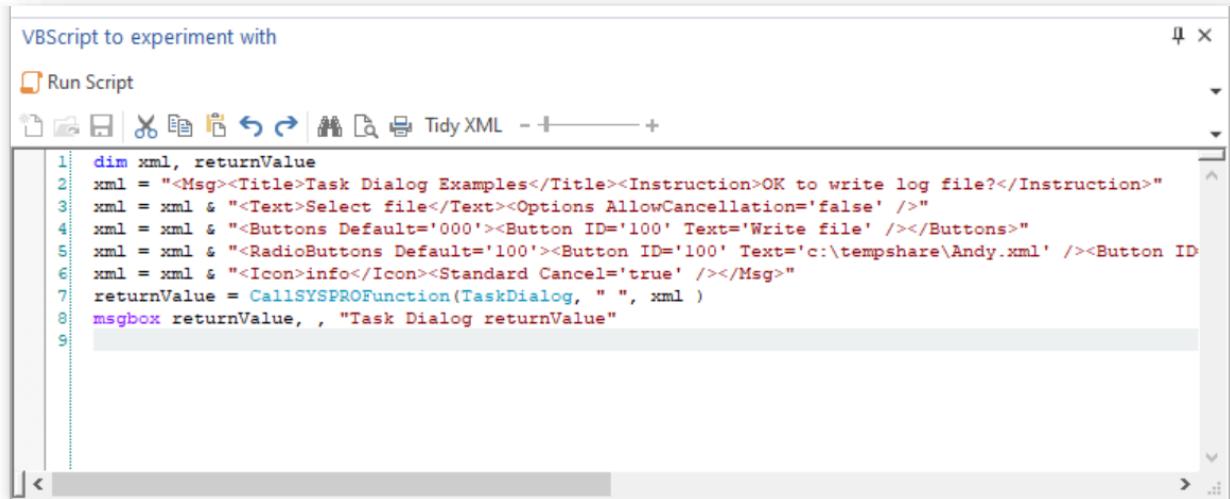
```
dim theme
theme ="vbscript"

dim script
script = "dim xml, returnValue" & vbcrlf
script = script & "xml = "" & tempstr & """ & vbcrlf
script = script & "returnValue = CallSYSPROFunction(TaskDialog, "", "", xml )" & vbcrlf
script = script & "msgbox returnValue, , ""Task Dialog returnValue"""& vbcrlf

call CallSYSPROFunction(SyntaxSetTheme, "SAMTSKS0", theme)

call CallSYSPROFunction(SyntaxInsertText, "SAMTSKS0", script)
```

Generates the following:



Please see the sample Application **SAMTSK** for full details.

XAML CONTROL

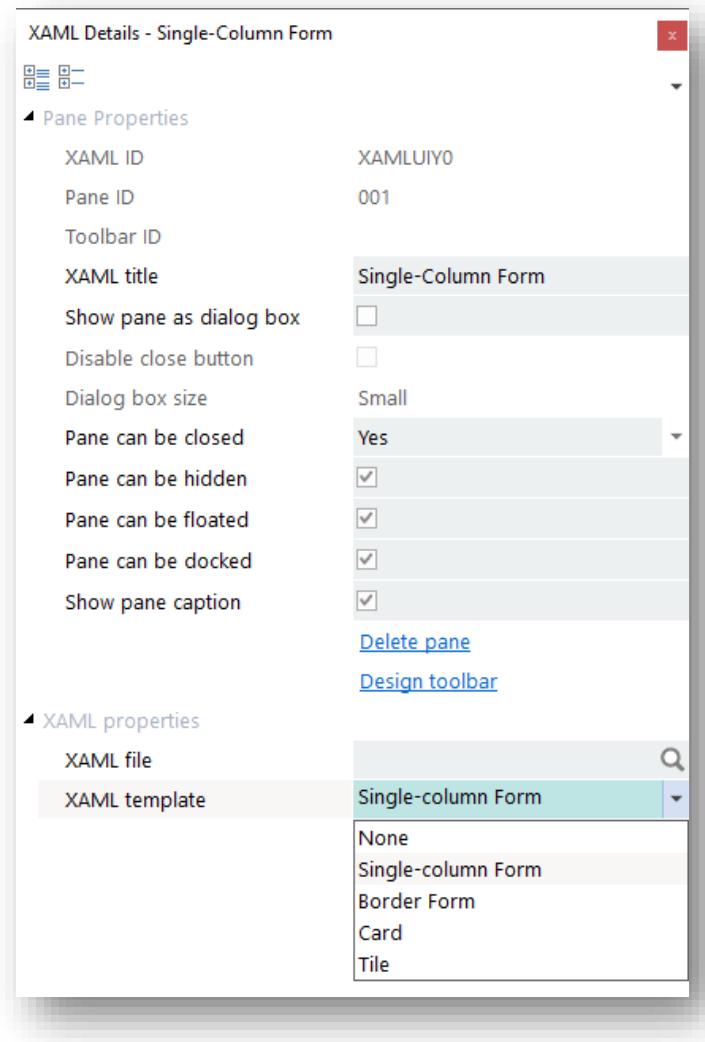
The XAML control used in the Application Designer to create a XAML Pane is a lightweight, fast control that uses a subset of the XAML language to create responsive layouts. It enables resizing of elements, graphical representation of data and gives the developer the ability to present information in many ways.

It can be used to create:

- SYSPRO Cards
- Tiles
- Forms
- Free format user interfaces

To simplify the creation of Cards, Tiles, and Forms in XAML, the Application Designer provides templates that can be populated with XML data.

These are Card, Tile, Single-column Form, and Border Form templates:



The Application Designer uses these templates and an XML string of data to create a XAML page for display. This removes all the complexity of creating the underlying XAML page, grids and styles in the VBScript and allows the developer to concentrate on providing data and content to the pane.

The Application Designer also links up SYSPRO key fields to the standard SYSPRO pop-up menus so that SYSPRO queries can be run from the form.

If no template is used, then the complete XAML Page must be constructed in your code and sent to the XAML control.

Note that XAML panes differ from traditional control type panes in one major area: The XAML page must be created in its entirety each time any changes are required and the XAML pane refreshed.

Please refer to the sample apps provided with the Application Designer for working examples.

THE XAML CALLOUTS

The following Callouts are provided for sending XAML data to the XAML control, or XML data to a template for the XAML to be built for you:

XAML callouts

- Show XAML content from a string (XAMLContent, "xxxxxxxx", XAMLstring) The string must contain valid XAML syntax.
- Show XAML content from a file (XAMLFile, "xxxxxxxx", filename) The file must contain XAML syntax.
- Update XAML template with values (XAMLValues, "xxxxxxxx", xmlString) The syntax for the XML string will depend on the XAML template in use.
General syntax:
<Form><Field Caption="General Information" Type="section" /><Field Caption="Balance" Value = "\$345,567.00" /></Form>

Defining the syntax of the XAML Page is crucial. If there are any errors in its construct, then the XAML control will simply not display anything.

Therefore, there are a couple of techniques that can be helpful:

- Use a XAML editor (such as **MICROSOFT EXPRESSION BLEND**) to construct a working prototype of the required result and make sure that what you are creating matches the working model.
Remember that the Application Designer uses a subset of XAML commands.
Please see the [Supported XAML tags](#) section for more detail.
- Write your XAML to a text file when you create it so that you can copy it into the XAML editor to see: a) if it works, and b) where any errors are.
There is a VBScript code snippet that can be used to do this.

If the basic XAML page works but occasionally it doesn't display anything, then this is typically caused by the data containing XML reserved characters in it (such as Ampersand (&) or Greater than sign (>)). These often occur in customer name or stock descriptions.

To avoid these, you should run the text through the **CheckXMLSpecialCharacters** code snippet which will replace these characters with ones acceptable to XML.

XAML EVENTS

A XAML control registers mouse click events for the following:

- Button
- Check box
- Radio button
- Hyperlinked
- TextBlock objects

Here's a list of the XAML syntax to register mouse click events for these objects, if you are hand-crafting XAML syntax:

- `<TextBlock MouseLeftButtonUp="MouseLeftButtonUpEvent" Tag="tagname">Hello</TextBlock>`
- `<Hyperlink Click="Hyperlink_Click" Tag="tagname">Hello</Hyperlink>`
- `<Button Click='Button_Click' Tag="tagname">Hello</Button>`
- `<CheckBox Click='CheckBox_Click' Tag="tagname">Hello</CheckBox>`
- `<RadioButton Click='RadioButton_Click' Tag="tagname" Content=' ' />`

Note the **Tag** element – this defines the value that will be returned to the **OnUserEvent** when any of these objects is clicked.

THE XAML FORM

To simplify the creation of a XAML type form, the Application Designer provides two basic templates.

The Single-column Form, which mimics the main SYSPRO style form data panes and the Border Form, which is the same single column structure but with borders around the fields.

Future releases of the Application Designer will contain further templates, but the operation will be the same.

Data being sent to the templates is of the following XML format:

```
dim xmlString
' For the CAPTION column, the MinWidth specifies minimum width and the MaxWidth
specifies maximum width.
xmlString = "<Form Heading='Customer Information: 0000001' >" &_
"<Field Caption='General Information' Type='section' />" &_
"<Field Caption='Balance' Value = '$ 345567.00' Type='Number' />" &_
"<Field Caption='Invoice amount' Value = '123' Type='Number' Decimals='2' />" &_
"<Field Caption='Customer' Value = '1' Tooltip='Acme Brush Company' />" &_
"<Field Caption='Stock code' Value = 'B100' Tooltip='Bicycle' />" &_
"<Field Caption='Email' Value = 'phil.duff@syspro.com' Type='hyperlink' />" &_
"<Field Caption=' ' Value = 'Show Invoices' Tag='TagHyperlink' Type='hyperlink' />" &_
"<Field Caption='Branch' Value = 'The East India Company, based in Singapore' />" &_
"<Field Caption='Salesperson' Value = 'James Robertson' Tooltip='Based in Seattle.' />" &_
"<Field Caption='Date last sale' Value = '2022-04-01' Type='Date' />" &_
"<Field Caption='Show Full Details' Type='button' Enabled='true' Tag='TagShowFulldetails' />" &_
"<Field Caption='Address Information' Type='section' />" &_
"<Field Caption='Street' Value = '1 Acme street' />" &_
"<Field Caption='Customer on hold' Value='1' Type='checkbox' Enabled='true' Tag='TagOnHold' />" &_
"<Field Caption='Profit percentage of overall sales' Value='30.56%' />" &_
"<Field Name='Set company date' Type='4' Value='2' List='For the company;Temporary;' Tag='TagRadio' />" &_
"<Field Caption='Address' Value='1st Avenue&#xA;Houghton&#xA;Johannesburg\n\nAfrica' + "/" &_
"<Field Caption='Building' Image='C:\avis-jpgs\chateau.jpg' Type='image' Stretch='Uniform' Width='250' Tag='TagImage' />" &_
"<Field Caption='Monthly sales' Type='linegraph' Title='Sales for B100' Value='12,23,45,67,100,105,4,0,7,45,50,4' Foreground='gray' Background='lemonchiffon' Tag='B100SalesLineGraph' Width='230' />" &
```

```

"<Field Caption='Monthly sales' Type='columngraph' Title='Sales for B100'
Value='12,23,45,67,100,105,4,0,7,45,50,4' Foreground='white'
Background='limegreen' Tag='B100SalesBarChart' Width='230' />" &_
"</Form>"

call CallSYSPROFunction(XAMLValues, "XAMLUIY0", xmlString)

```

Which will produce the following pane:

Single-Column Form

Customer Information: 0000001

General Information

Balance	\$ 345,567.00
Invoice amount	123.00
Customer	0000001
Stock code	B100
Email	phil.duff@syspro.com
Show Invoices	
Branch	The East India Company, based in Singapore
Salesperson	James Robertson
Date last sale	01/04/2022
Show Full Details	

Address Information

Street	1 Acme street
Customer on hold	<input checked="" type="checkbox"/>
Profit percentage of overall sales	30.56%
Set company date	<input type="radio"/> For the company <input checked="" type="radio"/> Temporary
Address	1st Avenue Houghton Johannesburg
Building	S Africa

Monthly sales

Sales for B10

12, 23, 45, 67, 100, 105, 4, 0, 7, 45, 50, 4

The visual styling, the type of the data field and what happens when they are clicked on are all controlled by attributes of the Form and Field elements.

XAML FORM – FORM AND FIELD ATTRIBUTES

These attributes are explained as follows:

Attribute	Explanation
<FORM> ATTRIBUTES	
Heading	This sets the form title.
MaxWidth	This sets the maximum width of the caption column. If the XAML form pane is resized, the caption column will not grow larger than this.
MinWidth	This sets the minimum width of the caption column. If the XAML form pane is resized, the caption column will not shrink smaller than this.
<FIELD> ATTRIBUTES	
Background	This uses a hex color code or a XAML standard color word to set the background of the Value attribute. The VALUE attribute corresponds to the data field. For graphs, this will set the color of the graph background.
Caption	This indicates the caption for the field. If the caption is recognized as a SYSPRO key field, then it will automatically be hyperlinked and linked to the pop-up menus of standard SYSPRO. To disable this behavior, use the Enabled='false' attribute (see below). If you don't want a caption next to a control (e.g. Buttons) then set it to a space: Caption = ' '
CaptionBackGround	This uses a hex color code or a XAML standard color word to set the caption background. Hex values are preceded with a hash sign (e.g. #FF0000 is the hex value for Red). Therefore, CaptionBackGround='#FF0000' and CaptionBackGround='Red' produce the same results.
CaptionForeGround	This uses a hex color code or a XAML standard color word to set caption text.
Decimals	This defines the number of decimals for a Number type field.
Enabled	This attribute is set to True by default as it enables hyperlinks, checkboxes, or buttons. Change this to False to disable.
Foreground	This uses a hex color code or a XAML standard color word to set value text. For graphs, this will set the color of the graph axis and data.
Image	This indicates a URL to an image file.

List	This indicates a semi colon list of options for the radio button control.
Stretch	<p>This defines how an image will fit into the image field.</p> <p>Values are:</p> <ul style="list-style-type: none"> ▪ Uniform The content is resized to fit the destination dimensions while it preserves its native aspect ratio. ▪ UniformToFill The content is resized to fit the destination dimensions while it preserves its native aspect ratio. If the aspect ratio of the destination rectangle differs from the source, the source content is clipped to fit the destination dimensions. ▪ Fill Content is resized to fill the destination dimensions. The aspect ratio is not preserved. ▪ None (default) The content preserves its original size.
Tag	This defines a return value if the element is clicked and is used with:
	<ul style="list-style-type: none"> ▪ Columngraph ▪ Linegraph ▪ Buttons ▪ Radio buttons ▪ Hyperlinks ▪ Check boxes ▪ TextBlock
Title	This is used to put a title on a graph.
Tooltip	This displays a tooltip when mouse hovers over the caption.
Type	<p>The Type attribute defines the type of field (e.g. Type='Date').</p> <p>The values are:</p> <ul style="list-style-type: none"> ▪ Section This defines a section. The CAPTION is displayed underlined. Any VALUE attribute is ignored. ▪ Date This will accept a date in the format CCYYMMDD or CCYY/MM/DD or CCYY-MM-DD. The date will be displayed in the same format as core SYSPRO. ▪ Number If the VALUE is preceded by a currency symbol, then this will be displayed. <ul style="list-style-type: none"> ▪ Numeric separators (commas and full stops) will be automatically applied. ▪ The number of decimals is defined by the DECIMALS attribute (see above).

	<ul style="list-style-type: none"> ▪ If the DECIMALS attribute is not set, then the number of decimals is determined from the VALUE string itself. ▪ Any leading currency sign will be preserved and displayed in front of the number unchanged. <p>▪ Button</p> <p>This defines a button.</p> <p>Button text is set by the CAPTION attribute.</p> <p>When clicked, it raises the XAMLButton event with param1 containing the TAG attribute (or the CAPTION if the TAG attribute has not been explicitly defined).</p> <p>Use the ENABLED attribute to enable and disable the button.</p> <p>▪ Hyperlink</p> <p>This defines a hyperlink.</p> <p>Hyperlink text is defined by the VALUE attribute.</p> <p>When clicked, it raises a XAMLLink event with param1 containing the TAG attribute (or the CAPTION if the TAG attribute has not been explicitly defined).</p> <p>If the VALUE is an email address, then the SYSPRO email interface is launched.</p> <p>For an email the event is not subsequently passed to the VBScript.</p> <p>▪ Image</p> <p>This defines an image field.</p> <p>The image size is controlled by STRETCH and WIDTH attributes.</p> <p>When clicked, it raises the XAMLLink event with param1 containing the TAG attribute (or the CAPTION if the TAG attribute has not been explicitly defined).</p> <p>▪ Checkbox</p> <p>This defines a checkbox.</p> <p>The check status is set by the VALUE attribute.</p> <p>0 = unchecked, 1 = checked.</p> <p>When clicked, it raises the XAMLCheckBox event with param1 containing the TAG attribute (or the CAPTION if the TAG attribute has not been explicitly defined) and param2 holds the check status; 0 for unchecked and 1 for checked.</p> <p>▪ Radio</p> <p>This defines a radio button.</p> <p>Options are defined in a LIST attribute (see above).</p> <p>The radio button options' position in the list are numbered from 1.</p> <p>A default option can be set by the VALUE attribute and a return value is defined in the TAG attribute.</p> <p>When an option is clicked, it raises the XAMLRadioButton event.</p>
--	---

	<p>The return value in param1 contains the TAG attribute (or the CAPTION if the TAG attribute has not been explicitly defined). Param 2 contains the position of the option in the list.</p> <ul style="list-style-type: none"> ▪ Linegraph This displays a line graph. Its values come from the Value attribute which should be a comma separated list of up to a maximum of 12 values. ▪ Columngraph This displays a column graph. Its values come from the VALUE attribute which should be a comma separated list of up to a maximum of 12 values. ▪ For graphs The x-axis labels are shown as 1,2,3,4,5,6,7,8,9,10,11,12, but can be overridden with the AxisLabels attribute which has the labels separated (e.g. AxisLabels='Jan, Feb, Apr ...'). These labels should not be longer than 4 or 5 characters. When a graph is clicked, it raises the XAMLButton event with param1 containing the TAG attribute (or the CAPTION if the TAG attribute has not been explicitly defined).
Width	<p>This defines the width of an image or a graph. The width and height of these controls is always the same. If not set, then the default is 100.</p>
Value	<p>This indicates the value to be displayed next to the caption. This is the data field displayed in the form. To insert a newline in a string, use the XAML syntax for a newline &#xA; or use a \n character.</p>

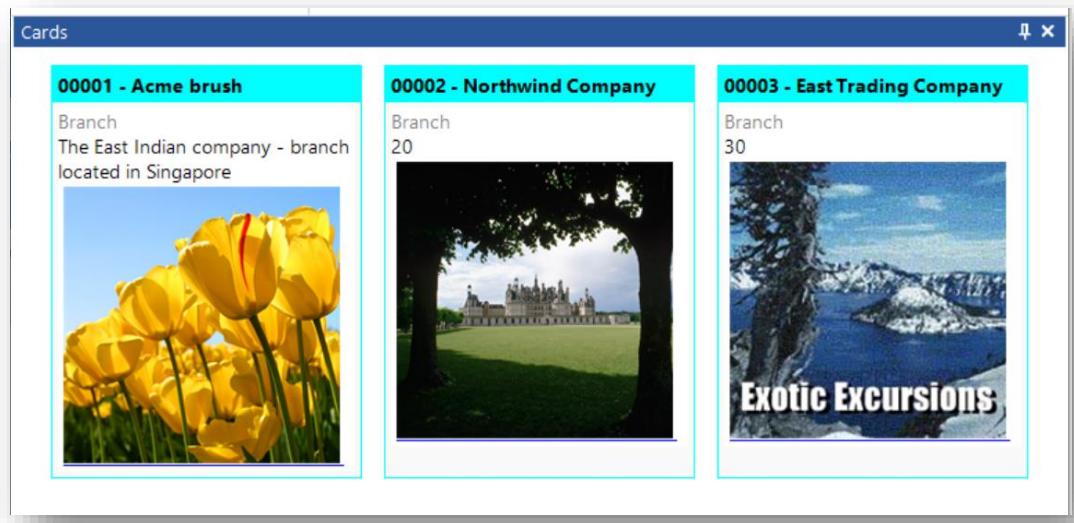
CARDS

The card template operates in a similar manner to Forms except that a new card is created for each **<Field>** element.

The following code:

```
dim xmlString
xmlString = "<Form>" &
"<Field Heading='00001 - Acme brush' Caption='Branch' Value = 'The East
Indian company - branch located in Singapore' Image='C:\avis-
jpgs\tulips.jpg' Tag='Tultips' />" &
"<Field Heading='00002 - Northwind Company' Caption='Branch' Value = '20'
Image='C:\avis-jpgs\chateau.jpg' Tag='Chateau' />" &
"<Field Heading='00003 - East Trading Company' Caption='Branch' Value =
'30' Image='C:\avis-jpgs\bayshore.jpg' Tag='Bayshore' />" &
"</Form>"
call CallSYSPROFunction(XAMLValues, "XAMLUIY1", xmlString)
```

Generates three cards as follows:



The pane is responsive, so the cards will move to accommodate the new size as the pane is resized.

CARDS – FIELD ATTRIBUTES

Attribute	Explanation
Heading	A heading at the top of the card.
Caption	A caption on the first line of the card.
Value	A data element displayed under the caption.
Image	A URL to an image.
Tag	Return value when image clicked.

When the image is clicked, it raises the **XAMLHyperlink** event and **Param1** will contain the value of **TAG**. If **TAG** is not defined, then param1 will be blank.

TILES

The tile template operates in the same way as a card. A single tile is in essence a giant button. Each **<Field>** element creates a new tile, and the pane is responsive enabling the tiles to re-organize to fit the available space.

Note: This TILE template is not to be confused with the Tiles that can be shown in a Tile Layout in an Avanti web view.

The following code:

```
dim xmlstring
xmlstring = "<Form>" &
"<Field Title='Customers on hold' Subtitle='Company: EDU1' Value='2' 
Footer='Demo for Phil' Foreground='White' Background='Green' Width='300' 
Tag='On Hold'" &
" Tooltip='&lt;Border&gt;&lt;Image Source=""file://c:\avis-jpgs\tulips.jpg"" 
Width=""200"" Height=""200"" Stretch=""UniformToFill""/&gt;&lt;/Border&gt;/'>" &
"<Field Title='Average days to pay' Subtitle=' ' Value='20' Footer='Days' 
Foreground='White' Background='LightBlue' Width='auto' Tag='Average Days' />" 
&
"<Field Type='linegraph' Title='Last 12 month sales' Subtitle=' ' 
Value='2045,25,67,4,6,7,8,9,10,6' Foreground='White' Background='Green' 
Width='auto' Tag='Sales by Month' />" &
"<Field Type='linegraph' Title='A few sales' Subtitle=' ' Value='12,24' 
Foreground='Black' Background='Linen' Tag='FewSales' Tooltip='Click to see 
details' />" &
"<Field Type='linegraph' Title='Sales for B100' 
Value='12,23,45,67,100,105,4,0,7,45,50,4' Foreground='Blue' 
Background='White' Tag='B100Sales' 
AxisLabels='Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec' />" &
"<Field Type='columngraph' Title='A100 - Sales Values last 12 Months' 
Value='12,23,45,67,100,105,4,0,7,45,50,4' Foreground='gray' 
Background='lemonchiffon' Tag='A100' />" &
"</Form>" 
call CallSYSPROFunction(XAMLValues, "XAMLUIY2", xmlString)
```

Generates the following tiles:



TILE – FIELD ATTRIBUTES

Attribute	Explanation
Title	The main heading at the top of the Tile.
Subtitle	A second (smaller) description below the title.
Value	A data element(s) depending on Type attribute (see below).
Footer	This is the text displayed at the bottom of the tile.
Foreground	<p>This uses a hex color code or a XAML standard color word to set the tile foreground (which is the text color of any titles, subtitles, footers, and values; or the color of graph lines and labels). Hex values are preceded with a hash sign (e.g. #FF0000 is the hex value for Red).</p> <p><i>For example:</i></p> <p>Foreground='#FF0000' and Foreground='Red' produce the same results.</p>
Background	This uses a hex color code or a XAML standard color word to set the tile background.
Width	<p>This specifies the width of the tile.</p> <p>Width='Auto' will size the tile to fit the data.</p>
Type	<p>This defines the type of control hosted by the tile:</p> <ul style="list-style-type: none"> ▪ Linegraph This displays a line graph. Its values come from the Value attribute which should be a comma separated list of up to a maximum of 12 values. ▪ Columngraph This displays a column graph. Its values come from the Value attribute which should be a comma separated list of up to a maximum of 12 values. ▪ For graphs The x-axis labels are shown as 1,2,3,4,5,6,7,8,9,10,11,12, but can be overridden with the AxisLabels attribute which has the labels separated (e.g. AxisLabels='Jan, Feb, Apr ... '). These labels should not be longer than 4 or 5 characters. When a graph is clicked, it raises the XAMLButton event with Param1 containing the TAG attribute (or the CAPTION if the TAG attribute has not been explicitly defined).
Tag	This is the return value when the tile is clicked.

Clicking on a tile will raise the **XAMLButton** event and the value of the **TAG** attribute is returned in **Param1**.

OTHER USES FOR XAML

XAML can be used in several controls, including:

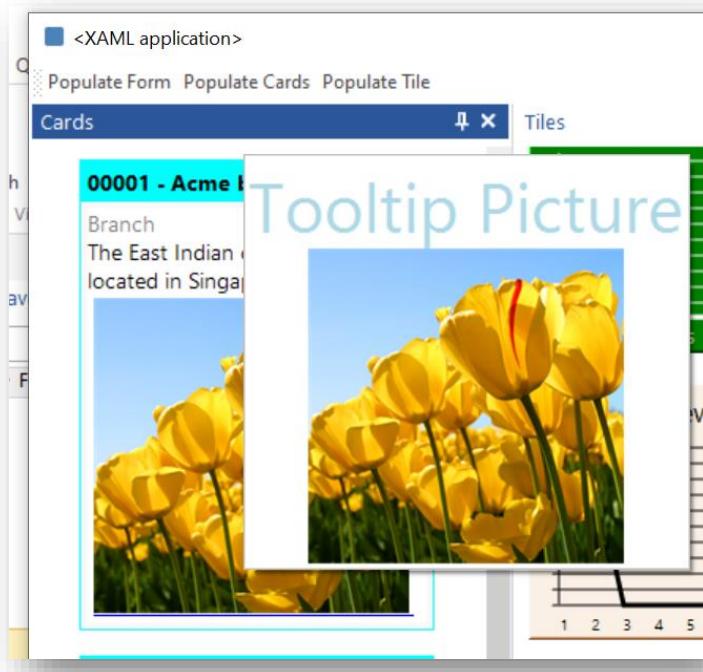
- Pane tooltips can be set by sending a string of XAML to a pane.
- Tooltips on card images can be set by providing XAML in the tooltip attribute.
- XAML can be used in tool tips for panes and tiles.
- It can also be used for a Tree view item.
- The XAML is processed in a grid control, so if the XAML you use works in a Grid then it will work in a tooltip.

XAML AND PANE TOOLTIPS

This code sets the tooltip for a pane to be an image:

```
call PopulateCard()
Tooltip = "<Border><StackPanel>" &_
    "<TextBlock Foreground='LightBlue' FontSize='43' FontFamily='Segoe UI' Text='Tooltip Picture' />" &_
    "<Image Source='file:///c:/avis-jpgs/tulips.jpg' Width='200' Height='200' Stretch='UniformToFill' />" &_
"</StackPanel></Border>"
call CallSYSPROFunction(PaneChangeTooltip, "XAMLU1Y1", Tooltip)
```

Hover the mouse over the pane title bar and the tooltip shows:



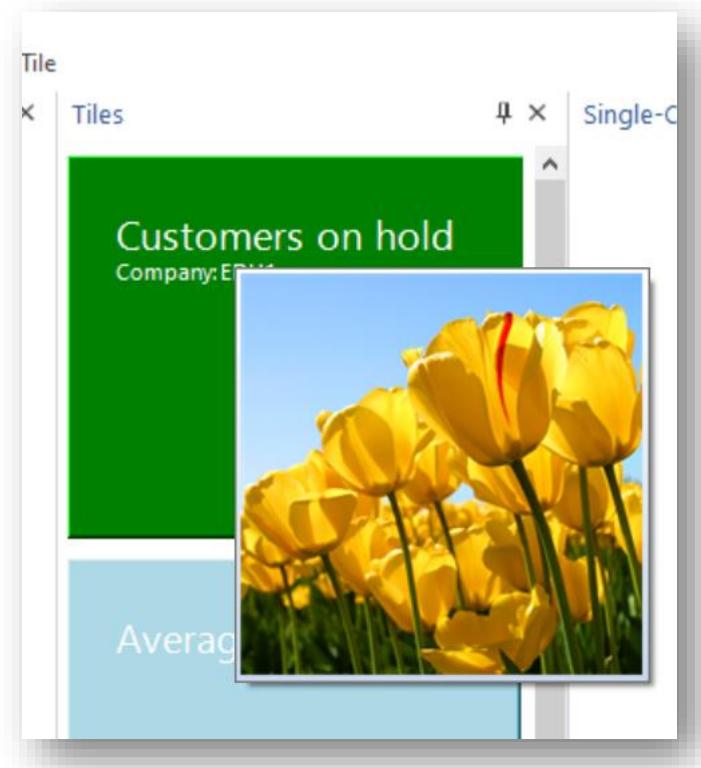
XAML AND CARD TOOLTIPS

The code below creates a tooltip on a tile:

```
& "<Field Title='Customers on hold' Subtitle='Company: EDU1' Value='2' Footer='Demo for Phil' " _  
& "Foreground='White' Background='Green' Width='300' Tag='On Hold' " _  
& "Tooltip='&lt;Border&gt;&lt;Image Source=""file://c:\avis-jpgs\tulips.jpg"" " _  
& "Width=""200"" Height=""200"" Stretch=""UniformToFill""/&gt;&lt;/Border&gt;/'>" _
```

Note the < and > characters have been replaced with XML acceptable codes **<** and **>**.

This could be used to show more information for the tile.



HANDLING XAML STRINGS

If provided as a string to an attribute for a control, you need to make sure that any XML reserved characters are replaced with their relevant encoding, otherwise XAML and XML get mixed up and nothing will work.

There is a code snippet provided in the Application Designer called

CheckXMLSpecialCharacters which will parse a string and substitute the relevant codes if required and of course you can hard code the string.

SUPPORTED XAML TAGS

The following is a list of supported XAML tags and their attributes:

XAML tag	Attributes	
Page	<ul style="list-style-type: none"> ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width 	<ul style="list-style-type: none"> ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ MaxHeight
Border	<ul style="list-style-type: none"> ▪ Cursor ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width ▪ Height 	<ul style="list-style-type: none"> ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ MaxHeight ▪ Padding ▪ Background ▪ BorderThickness ▪ BorderBrush
TextBlock	<ul style="list-style-type: none"> ▪ Cursor ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ MaxHeight 	<ul style="list-style-type: none"> ▪ Padding ▪ Background ▪ Foreground ▪ TextWrapping ▪ TextAlignement ▪ TextDecorations ▪ FontSize ▪ FontWeight ▪ FontFamily ▪ FontStyle ▪ BaselineAlignment ▪ Text
Image	<ul style="list-style-type: none"> ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width 	<ul style="list-style-type: none"> ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ MaxHeight ▪ Source
Run	<ul style="list-style-type: none"> ▪ Cursor ▪ Background ▪ Foreground ▪ TextDecorations ▪ FontSize 	<ul style="list-style-type: none"> ▪ FontWeight ▪ FontFamily ▪ FontStyle ▪ BaselineAlignment ▪ Text
LineBreak		

XAML tag	Attributes	
Span	<ul style="list-style-type: none"> ▪ Background ▪ Foreground ▪ TextDecorations ▪ FontSize 	<ul style="list-style-type: none"> ▪ FontWeight ▪ FontFamily ▪ FontStyle
Bold	<ul style="list-style-type: none"> ▪ Background ▪ Foreground ▪ TextDecorations ▪ FontSize 	<ul style="list-style-type: none"> ▪ FontWeight ▪ FontFamily ▪ FontStyle
Italic	<ul style="list-style-type: none"> ▪ Background ▪ Foreground ▪ TextDecorations ▪ FontSize 	<ul style="list-style-type: none"> ▪ FontWeight ▪ FontFamily ▪ FontStyle
Underline	<ul style="list-style-type: none"> ▪ Background ▪ Foreground ▪ TextDecorations ▪ FontSize 	<ul style="list-style-type: none"> ▪ FontWeight ▪ FontFamily ▪ FontStyle
InlineUIContainer	<ul style="list-style-type: none"> ▪ Background ▪ Foreground ▪ TextDecorations ▪ FontSize 	<ul style="list-style-type: none"> ▪ FontWeight ▪ FontFamily ▪ FontStyle
Hyperlink	<ul style="list-style-type: none"> ▪ Background ▪ Foreground ▪ TextDecorations ▪ FontSize 	<ul style="list-style-type: none"> ▪ FontWeight ▪ FontFamily ▪ FontStyle
Panel	<ul style="list-style-type: none"> ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width 	<ul style="list-style-type: none"> ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ MaxHeight ▪ Background
StackPanel	<ul style="list-style-type: none"> ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width ▪ Height 	<ul style="list-style-type: none"> ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ MaxHeight ▪ Background ▪ Orientation

XAML tag	Attributes	
WrapPanel	<ul style="list-style-type: none"> ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width ▪ Height ▪ MinWidth 	<ul style="list-style-type: none"> ▪ MaxWidth ▪ MinHeight ▪ MaxHeight ▪ Background ▪ Orientation ▪ ItemHeight ▪ ItemWidth
Canvas	<ul style="list-style-type: none"> ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width ▪ Height ▪ MinWidth 	<ul style="list-style-type: none"> ▪ MaxWidth ▪ MinHeight ▪ MaxHeight ▪ Background ▪ Left ▪ Right ▪ Top ▪ Bottom
Grid	<ul style="list-style-type: none"> ▪ Style ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width ▪ Height 	<ul style="list-style-type: none"> ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ MaxHeight ▪ Background ▪ ColumnDefinitions ▪ RowDefinitions
ColumnDefinition	<ul style="list-style-type: none"> ▪ Width ▪ MinWidth 	<ul style="list-style-type: none"> ▪ MaxWidth
RowDefinition	<ul style="list-style-type: none"> ▪ Height ▪ MinHeight 	<ul style="list-style-type: none"> ▪ MaxHeight
Polygon	<ul style="list-style-type: none"> ▪ Width ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width 	<ul style="list-style-type: none"> ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ Points
Polyline	<ul style="list-style-type: none"> ▪ Width ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width 	<ul style="list-style-type: none"> ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ Points

XAML tag	Attributes	
Rectangle	<ul style="list-style-type: none"> ▪ IsMouseOver ▪ Width ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment 	<ul style="list-style-type: none"> ▪ Width ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight
Ellipse	<ul style="list-style-type: none"> ▪ Width ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment 	<ul style="list-style-type: none"> ▪ Width ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight
Line	<ul style="list-style-type: none"> ▪ Width ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment ▪ Width 	<ul style="list-style-type: none"> ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ X1, Y1, X2, Y2
Button	<ul style="list-style-type: none"> ▪ IsPressed ▪ Style ▪ Width ▪ Margin ▪ ClipToBounds ▪ HorizontalAlignment ▪ VerticalAlignment 	<ul style="list-style-type: none"> ▪ Width ▪ Height ▪ MinWidth ▪ MaxWidth ▪ MinHeight ▪ Content
CheckBox		
RadioButton		
ScrollViewer	<ul style="list-style-type: none"> ▪ Style ▪ VerticalScrollBarVisibility 	<ul style="list-style-type: none"> ▪ HorizontalScrollBarVisibility
SolidColorBrush	<ul style="list-style-type: none"> ▪ Color 	
LinearGradientBrush	<ul style="list-style-type: none"> ▪ StartPoint ▪ EndPoint 	<ul style="list-style-type: none"> ▪ GradientStops
GradientStop	<ul style="list-style-type: none"> ▪ Color 	<ul style="list-style-type: none"> ▪ Offset
Style	<ul style="list-style-type: none"> ▪ TargetType ▪ BasedOn 	<ul style="list-style-type: none"> ▪ Resources
Setter	<ul style="list-style-type: none"> ▪ Property 	<ul style="list-style-type: none"> ▪ Value
Events	<ul style="list-style-type: none"> ▪ MouseEnter ▪ MouseLeave ▪MouseMove ▪ MouseLeftButtonUp 	<ul style="list-style-type: none"> ▪ MouseLeftButtonDown ▪ MouseRightButtonUp ▪ MouseRightButtonDown

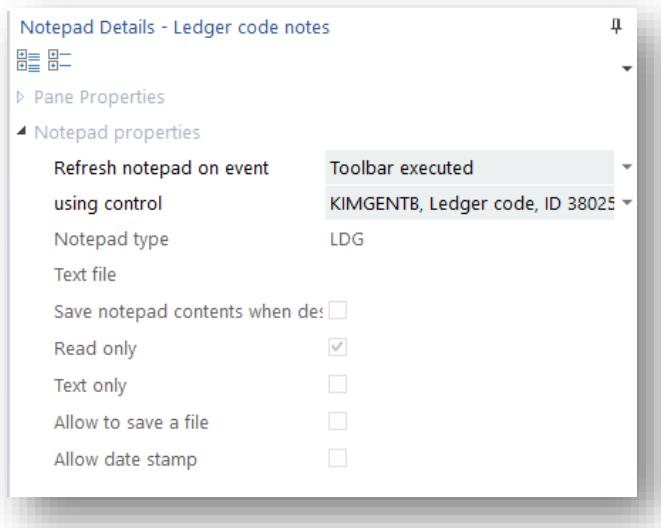
NOTEPAD CONTROL

The notepad control allows you view or modify RTF (rich-text format) or Text notes. The control can be used to show notes that are embedded in the application (perhaps to show HELP text) and, more typically, can be used to show notes linked to KEY fields such as a Customer or Stock code.

The contents of a **Notepad control** may be automatically refreshed when the value of a toolbar control changes. This saves having to provide logic in your script to read for the RTF Notes when entering, for example, a Customer or Ledger code.

In the option **Refresh notepad on event** select **Toolbar executed**, then select the toolbar control that will act as the KEY to refresh the notepad contents.

Based on the selected toolbar, the App Designer will indicate the correct Notepad type:



When the selected toolbar control value is changed then the notes will be automatically refreshed in the notepad control.

If the notes cannot be accessed, then the notepad contents will be cleared.

Note that this can only be used to show Notes – this technique cannot be used to save notes.

There are a number of callouts that may be used to manage the notepad document:

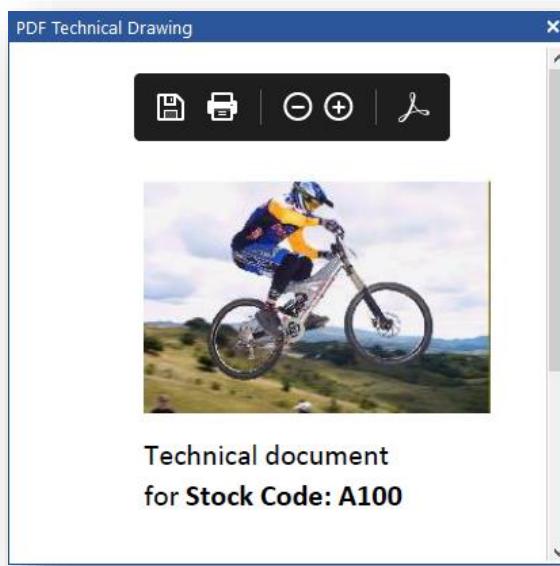
Notepad callouts	
Open a notepad file	(NotepadOpenFile, "xxxxxxxx", filename)
Open a notepad file for editing	(NotepadOpenFileEdit, "xxxxxxxx", filename)
Replace notepad text	(NotepadReplaceText, "xxxxxxxx", string) This will replace all the text in the notepad with the text string.
Append notepad text	(NotepadAppendText, "xxxxxxxx", string) This will append the text string to the notepad.
Get notepad text	(NotepadGetText, "xxxxxxxx", " ") Returns the notepad text. The return value can be used in a notepad business object.
Set focus to the notepad	(NotepadSetFocus, "xxxxxxxx", " ")
Save the notepad to a file	(NotepadSaveFile, "xxxxxxxx", filename)
Clear the notepad	(NotepadClear, "xxxxxxxx", " ")
Set the notepad to be read-only	(NotepadReadOnly, "xxxxxxxx", " ")
Set the notepad to be read-write	(NotepadReadWrite, "xxxxxxxx", " ")
Checks if the notepad changed	(NotepadCheckForChanges, "xxxxxxxx", " ") Returns 0 (no changes) or 1 (text has changed).

DOCUMENT VIEWER

The document viewer control allows you to view just about any type of document (.PDF, .JPG, .XLSX, .DOCX, .PPTX, .GIF, .PNG, .XAML, .TXT, .TIFF, .MSG, .BMP, and many others).

For example:

If you specify a PDF document to view, then the document will be rendered:



Similarly, pointing the viewer to a XML file will render it like this:



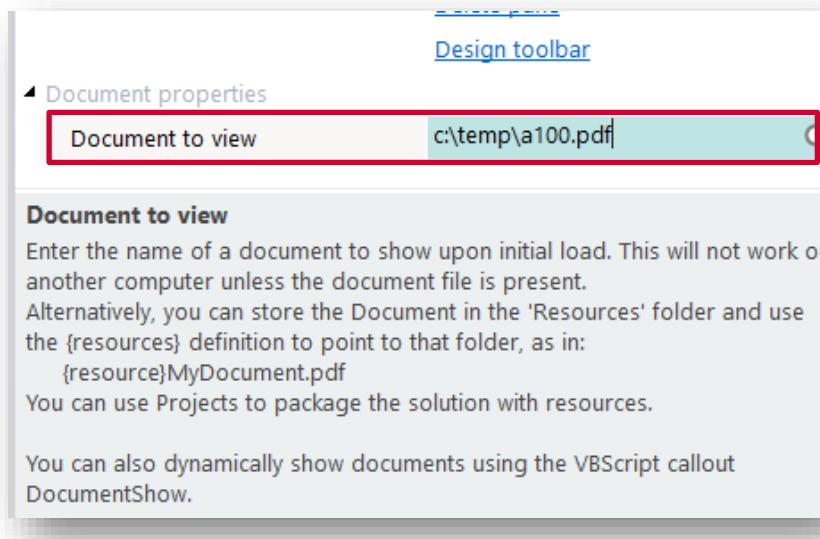
The screenshot shows a window titled "PDF Technical Drawing". Inside, there is a code editor displaying an XML document. The XML is color-coded for syntax highlighting. It includes declarations, a query section with a table name, return rows, columns (with specific column names), and a where clause containing an expression with an open bracket, column name, condition (EQ), value (ArCustomer), and close bracket.

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<!-- Copyright 1994-2022 SYSPRO Ltd.-->
<!-- Sample XML for the Generic find Business Object -->
- <Query xsd:noNamespaceSchemaLocation="COMFND.XSD"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance">
  <TableName>DdsColumns</TableName>
  <ReturnRows>1000</ReturnRows>
  - <Columns>
    <Column>ColumnName</Column>
    <Column>Description</Column>
    <Column>AlphaNumericDate</Column>
    <Column>Length</Column>
  </Columns>
  - <Where>
    - <Expression>
      <OpenBracket>(</OpenBracket>
      <Column>TableName</Column>
      <Condition>EQ</Condition>
      <Value>ArCustomer</Value>
      <CloseBracket>)</CloseBracket>
    </Expression>
  </Where>
</Query>
```

DEFINING THE DOCUMENT TO VIEW

There are two ways of specifying the name of a document to view:

1. In the document properties:



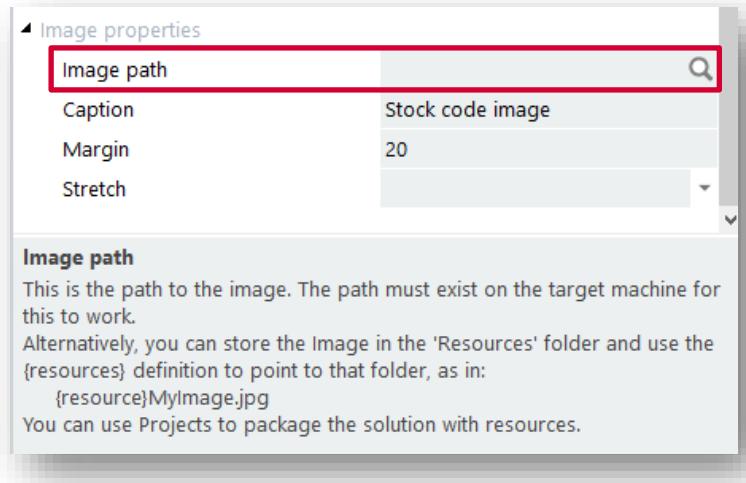
2. Using the **DocumentShow** VBScript callout to dynamically change the content:

```
dim filename
filename = "C:\Temp\Demo Documents\al100.pdf"
call CallSYSPROFunction(DocumentShow, "NOTE00P0", filename)
```

IMAGE VIEWER

There are two ways of displaying images in the control:

1. By specifying a path to the image in the properties pane:



2. Or by using the VBScript callout **ImageRefresh** to dynamically change the image:

```
call CallSYSPROFunction(ImageRefresh, "NOTE0010", "c:\avis-jpgs\chateau.jpg")
```



Optionally, you can also specify a caption - either in the **Properties** window or dynamically using the **ImageCaption** callout:

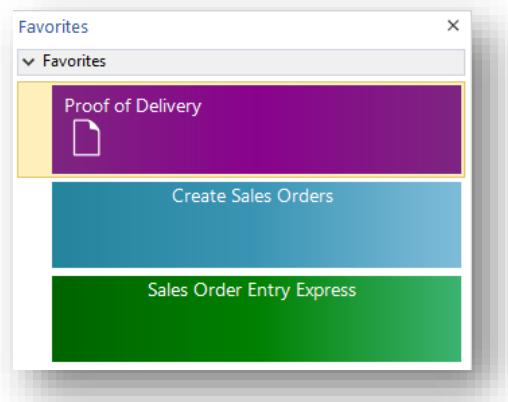
```
call CallSYSPROFunction(ImageCaption, "NOTE0010", "new image caption")
```

TILE CONTROL

The Tile control is the same control that is used for the **Favourites** pane in SYSPRO's main menu and is primarily used for navigation content.

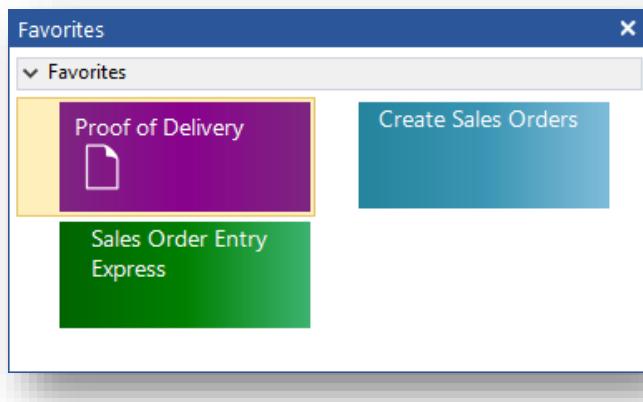
Each tile can contain content (a title, a tooltip, and an image) with various attributes (style, colors, icons) and can launch a SYSPRO program or executable. Tiles can also be grouped into **Categories**.

Here's an example of some tiles:



Tiles can be stretched across the control area (as in the above example) or can be displayed in multiple columns.

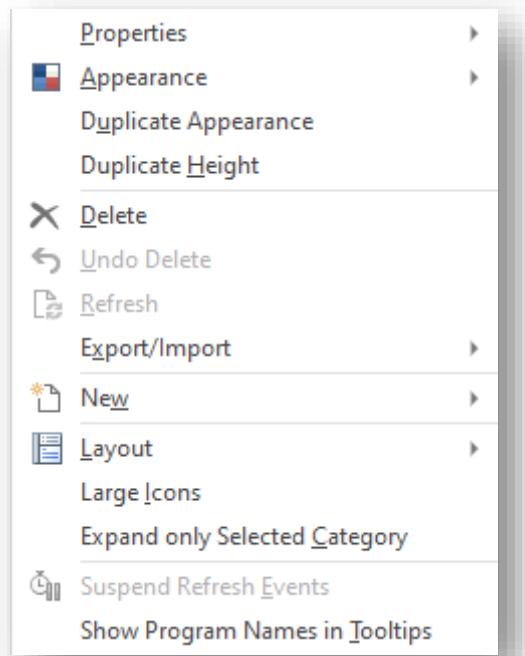
Here's an example of the same tiles but with **Multi-column groups** switched on.



DESIGNING TILES

To add a tile to the control, proceed as follows:

1. Drag a program from the **Programs List** or **Recent Programs** list in the main menu and drop it on the Tile control.
2. Drag a tile from your **Favourites** menu items onto the Tile control.
3. Drag any file from Windows Explorer onto the Tile control.
4. Right-click on the Tile control and a popup menu will appear:



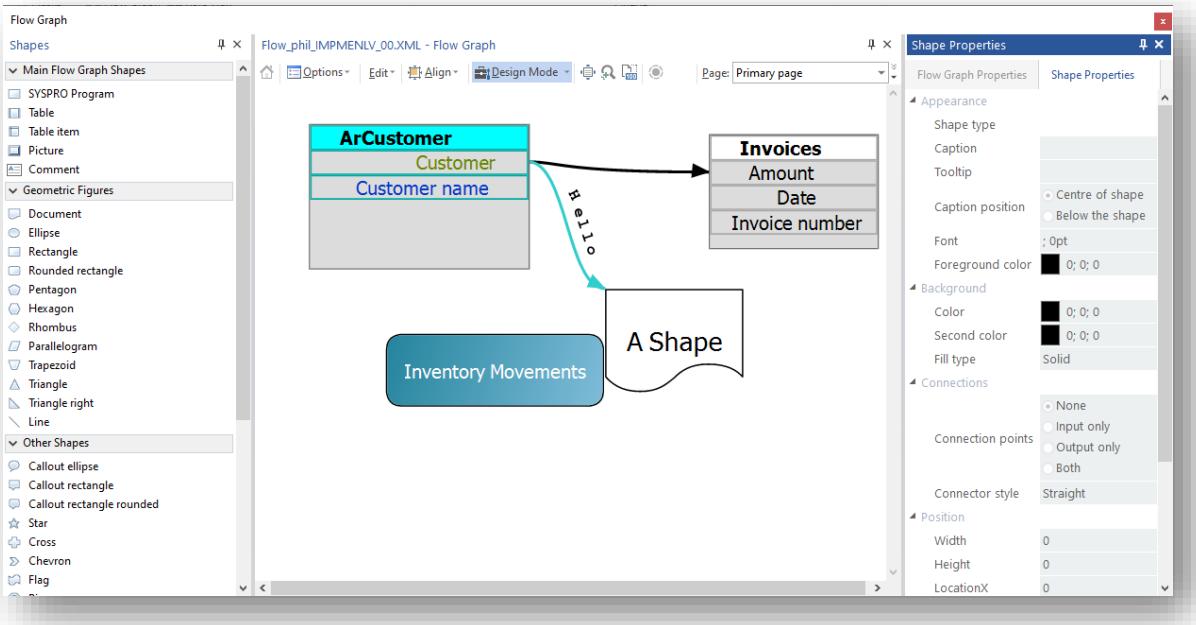
From here you can create a new Tile, delete a tile, add a category, and change various styling options.

FLOWGRAPH CONTROL

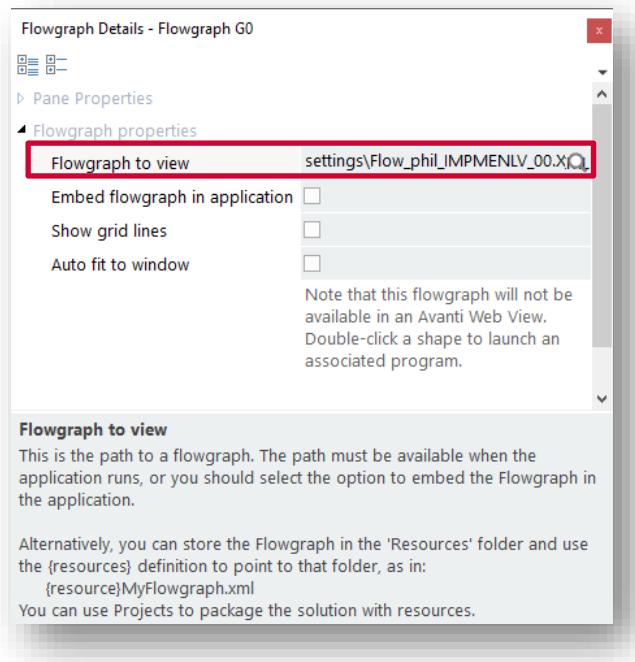
The flowgraph control can view any existing Flowgraph (that was created in the Flowgraph designer) found in the SYSPRO main menu.

Flowgraphs are a great way of showing processes to a user, especially since clicking on a 'shape' can launch any SYSPRO program or executable.

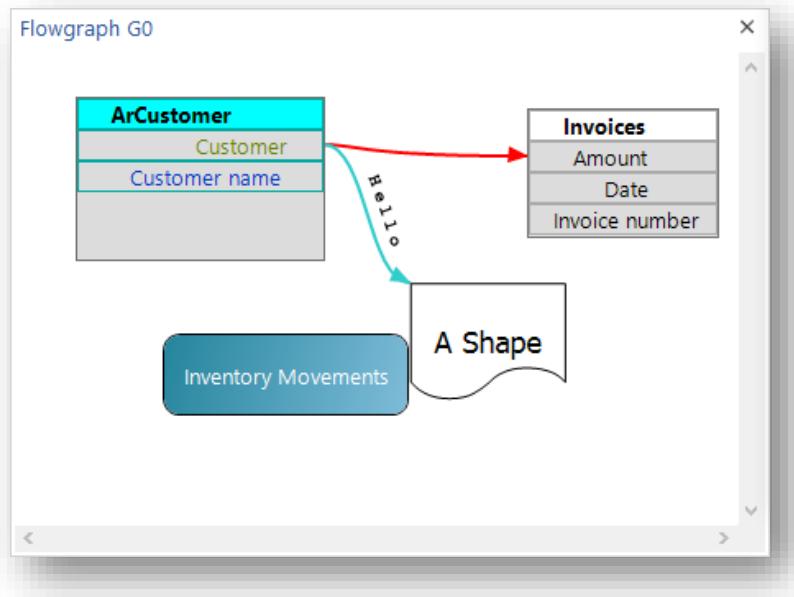
You should use the SYSPRO Flowgraph designer to create your flowchart:



and then specify the path to the Flowgraph in the **Flowgraph to view** property:



The flowgraph will then be displayed in the control:



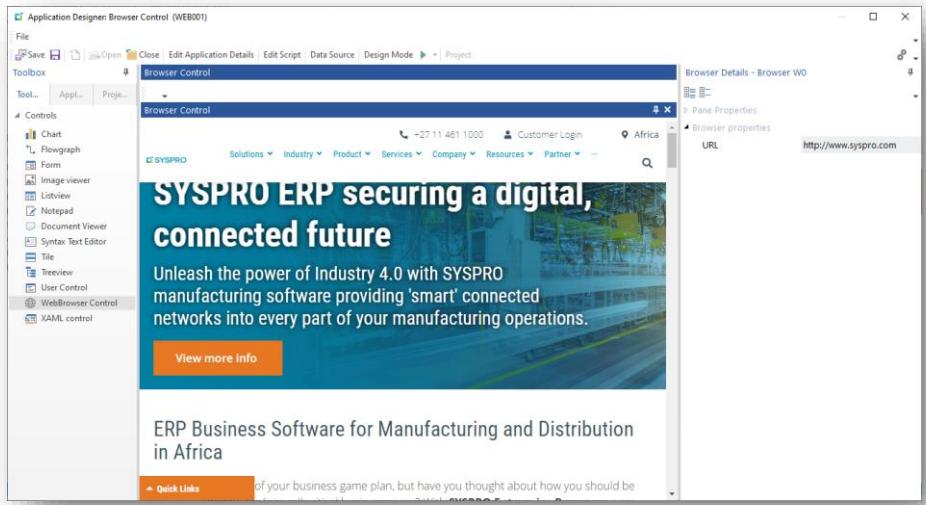
You should consider enabling the **Embed flowgraph in the application** option, as this option ensures that whatever flowgraph is specified, the contents are embedded in the application text file when the application is saved.

You can then deploy the application to the target machine knowing that the Flowgraph will work on that machine. If you do not embed the flowgraph in the application, the flowgraph path must exist on the target machine.

Note: The flowgraph control cannot be used in Avanti.

WEBBROWSER CONTROL

The browser control can navigate to any specified URL:



Or you can use VBScript callouts to dynamically navigate to any URL.

Here are the available callouts:

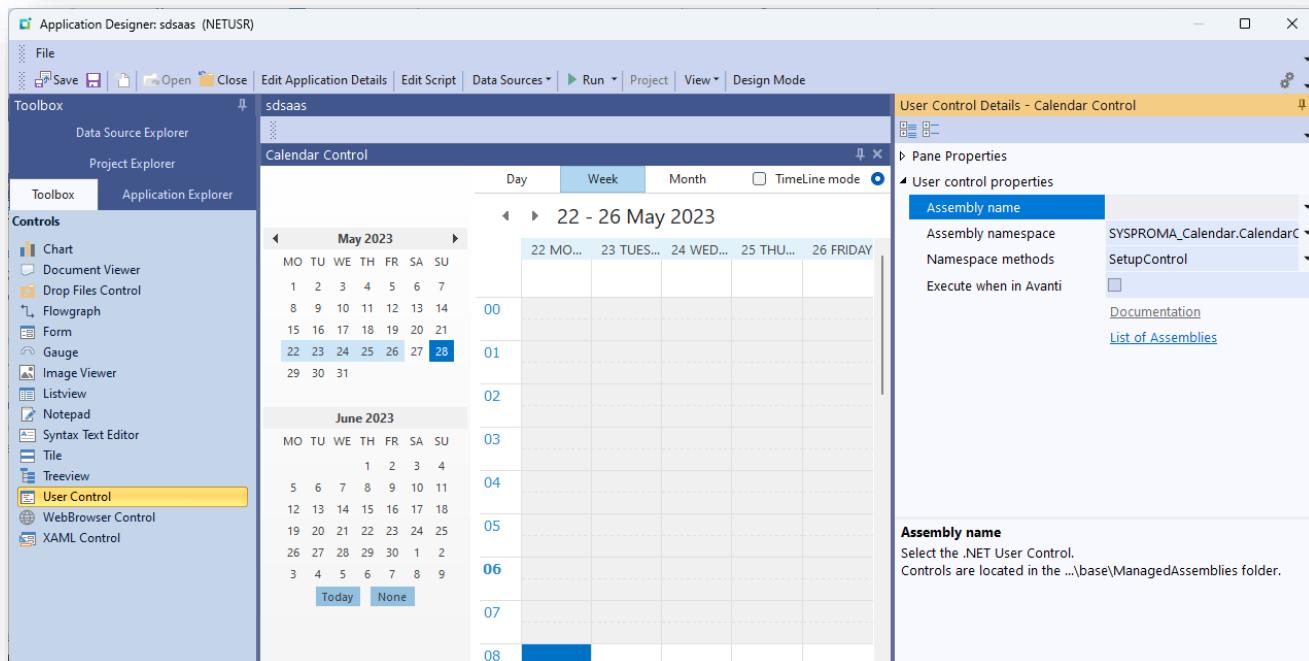
4 Browser callouts	
Navigate to a web address	(WebAddress, 'xxxxxxxx', 'URL')
Get web address	(WebGetAddress, 'xxxxxxxx', '')
Get web name	(WebGetName, 'xxxxxxxx', '')
Go back	(WebBack, 'xxxxxxxx', '')
Go forward	(WebForward, 'xxxxxxxx', '')
Refresh web page	(WebRefresh, 'xxxxxxxx', '')
	The web address must be valid.
	Returns the current URL.
	Returns the current name.
	During a browsing session, the WebBrowser control maintains a history of Web sites visited during a session.
	During a browsing session, the WebBrowser control maintains a history of Web sites visited during a session.
	Reloads the web page.

Note: The web browser control will use the latest **MICROSOFT** browser technology that's available on the computer.

If **MICROSOFT EDGE WEBVIEW2** has been installed, for example, then that will be used to render content.

USER CONTROL

For more advanced users, you can create .NET User Controls using Visual Studio and then use them in the Application Designer.



Note: The creation of such a control is beyond the scope of this document).

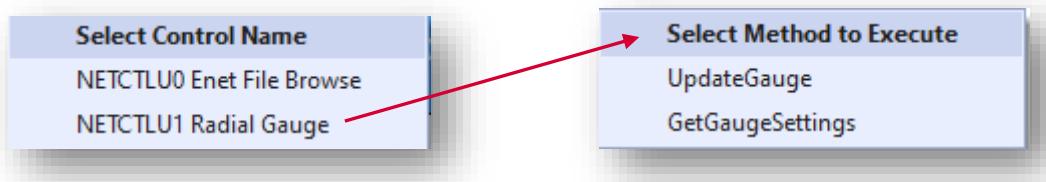
Within the application's script you can execute a method on a User Control using the callout **UserExecuteMethod**. The script syntax looks like this:

```
CallSYSPROFunction(UserExecuteMethod, "xxxxxxxx", "methodName|param1|param2")
```

where **xxxxxxxx** is the name of the control within the application, and Param1 and Param2 are optional parameters that can be passed to the control when executing the method.

User Control callouts	
Execute a user control method	(UserExecuteMethod, "xxxxxxxx", "methodName Param1 Param2") Requires the method to execute and optionally 2 parameters - separated by character. The available methods will be shown in a popup after selecting a control.

When double-clicking on **Execute a user control method** the available User Control names will be shown in a popup, and when you select a user control the available methods in that control will be shown:



If the user control returns a value, then you can extract it using this syntax:

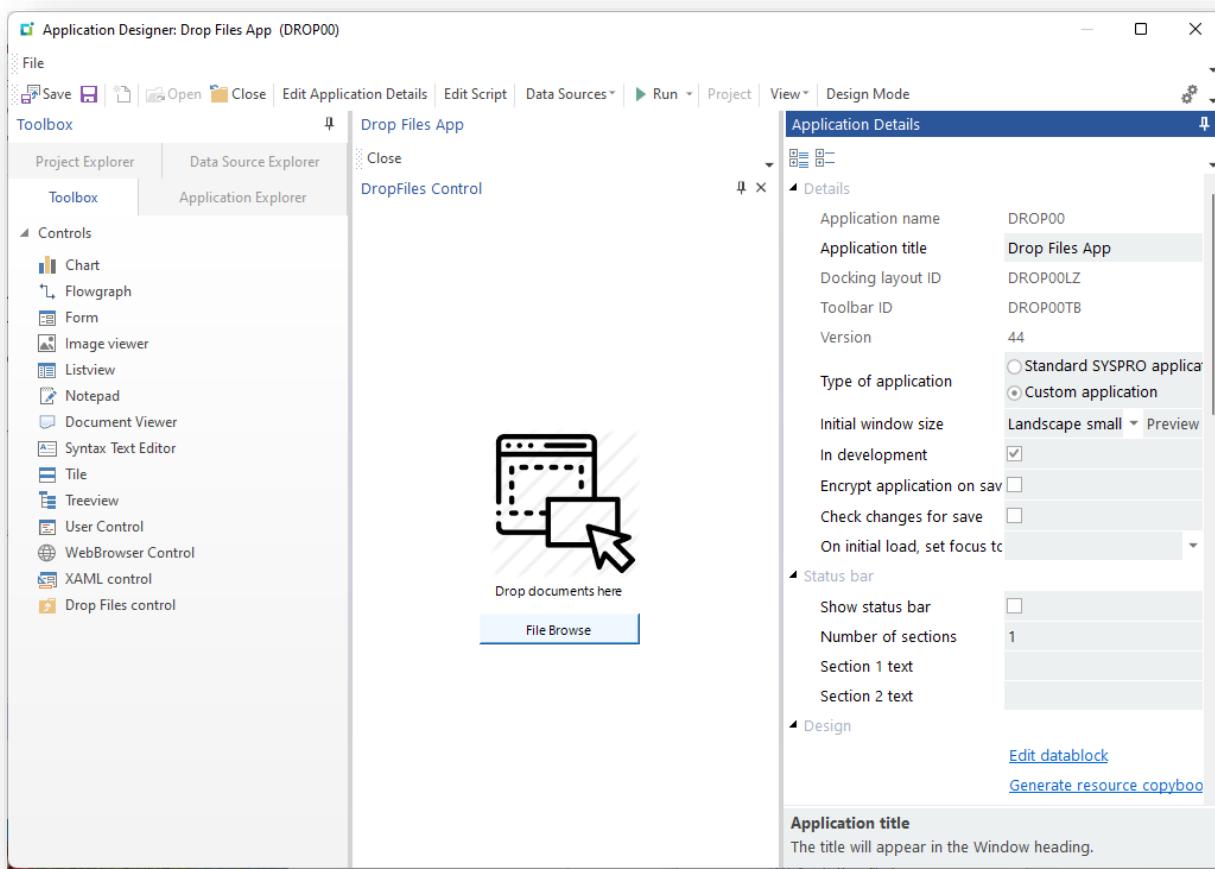
```
returnValue = CallSYSPROFunction(UserExecuteMethod, "NETCTLU0",  
"FileBrowse|*.xml")
```

DROP FILES CONTROL

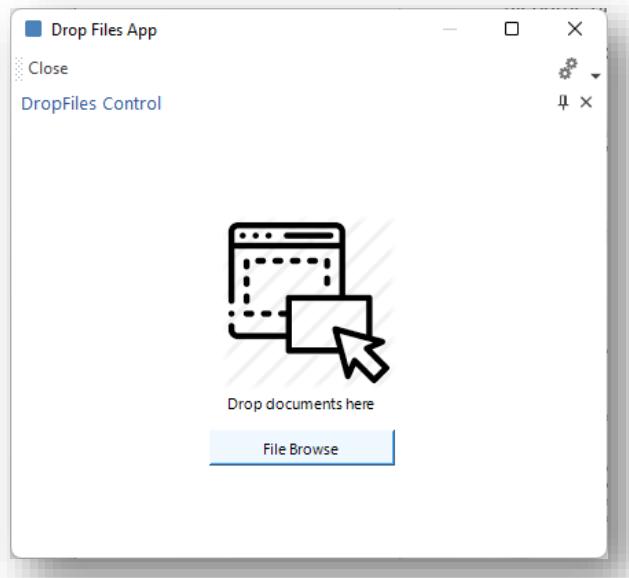
This control allows files to be dragged from Windows Explorer onto a document control area.

You can use this control to manage documents and files that your application needs to process. This might alleviate the need to have a file browse button on a form to process a filename one at a time.

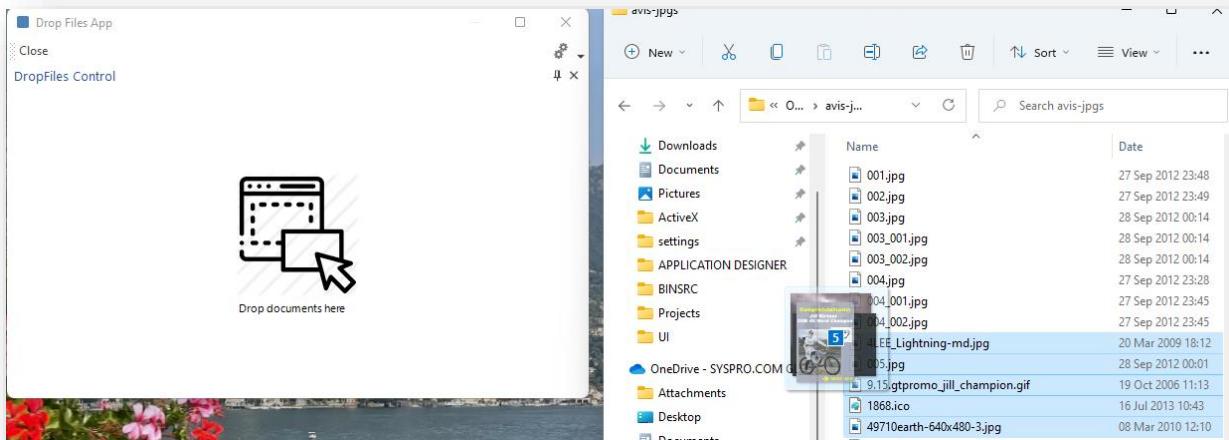
In your application, click on **Drop Files control** and a pane will be added to your application like this:



Now run the application, which will look like this:

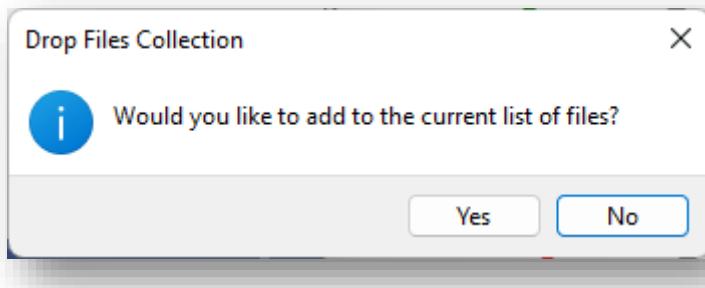


Now select and drag one or more files from Windows Explorer onto the drop area of the control:



Alternatively, you can click on **File Browse** to select files by browsing the file system.

If you subsequently drop more files onto the control, you will then be prompted:



This is particularly useful if you wish to create a list from more than one folder.

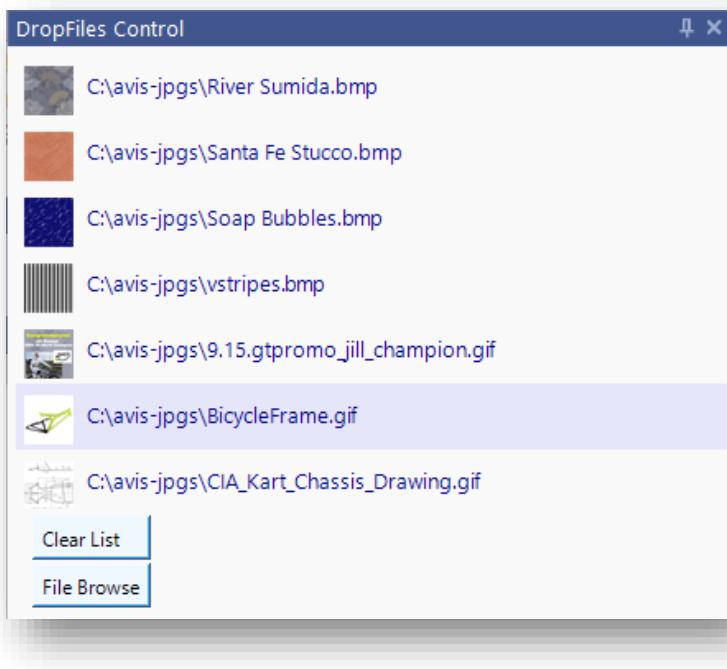
When one or more files are dragged and dropped onto the control area the script event **DroppedFilesEvent** will be raised in the **AppDesigner_OnUserEvent**. The list of files dropped onto the control will be passed into **Param1**, each filename separated by a TAB character.

Here's some example code to process the list of dropped files:

```
' Dropped files event. Param1 contains a list of filenames, each separated by
a TAB character
Function
AppDesigner_OnUserEvent(EventType, EventName, EventID, Param1, Param2, Param3)
if eventType = DroppedFilesEvent and eventID = DroppedFiles then
    dim FileNameArray, x1, ArraySize
    FileNameArray = split(Param1, Chr(10))
    ArraySize = ubound(FileNameArray)
    for x1 = 0 to ArraySize
        msgbox "Filename: " + FileNameArray(x1)
    next
end if
```

Optionally, you can also show the list of dropped files in the control area.

When you click on any of the listed files, the **DroppedFileSelected** script event is triggered and **Param1** will contain the name of the file:



You can also change the **Number of columns to show** in this list.

This is what the list will look like with 2 columns to show:



Finally, you can change the Image size to be Normal, Medium or Large. This property applies to files that are of type image.

The list of files can be cleared by clicking on the **Clear List** button.

You can also the script callout **DropAddFiles** to add a collection of filenames to the drop control:

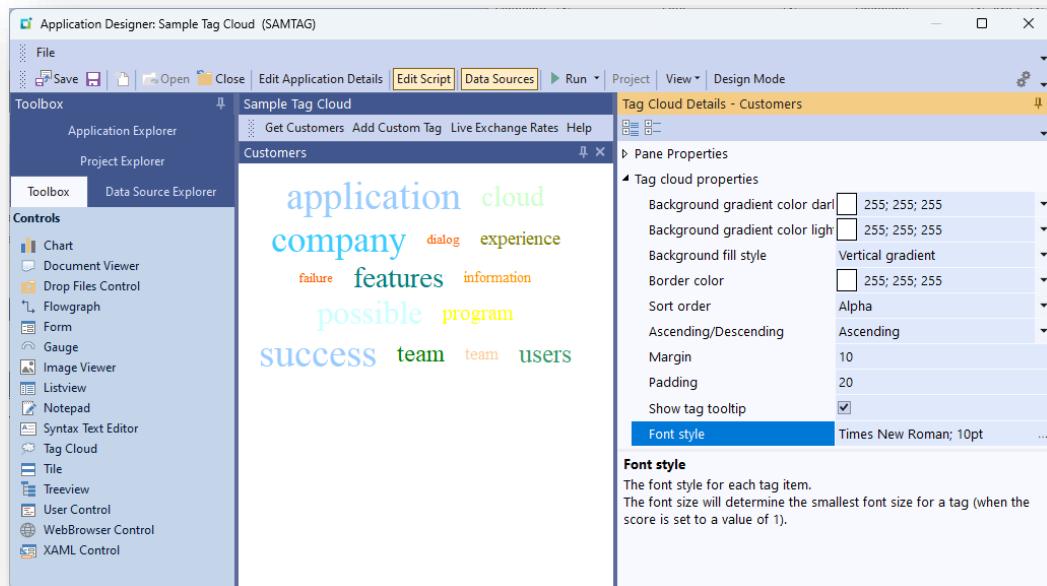
▲ Drop Control callouts

Add filenames to list (DropAddFiles, "xxxxxxxx", "list of files") Requires a list of filenames separated by the ; character.

Note: When clicking on a file the **DroppedFileSelected** script event is triggered.

TAG CLOUD CONTROL

The Tag Cloud control shows text values ranked by font size according to a score – the higher the score for an item, the larger the font size.



Apart from the properties for the control, the contents of the Tag Cloud control are controlled by a number of script callouts:

Tag cloud callouts

Add tag cloud items

(TagCloudAddItems, "xxxxxxxx", xmlstring)

xmlString syntax (in the same format as a "list" custom data source):

```
<Query> <Row>
<TagText>xxxx</TagText>
<TagScore>xxxx</TagScore>
</Row> </Query>
```

Clear tag cloud and populate with new items

(TagCloudClearAndPopulate, "xxxxxxxx", xmlstring)

xmlString syntax (in the same format as a "list" custom data source):

```
<Query> <Row>
<TagText>xxxx</TagText>
<TagScore>xxxx</TagScore>
</Row> </Query>
```

Clear tag cloud

(TagCloudClear, "xxxxxxxx", "")

The sample application **SAMTAG** demonstrates how to use the Tag Cloud control.

Note that there is no equivalent control yet for Avanti.

Working with Projects

Projects can help manage multiple applications and resources, especially if deploying a full solution to an end-user. Projects can also be useful simply to keep a track of applications that you are working on.

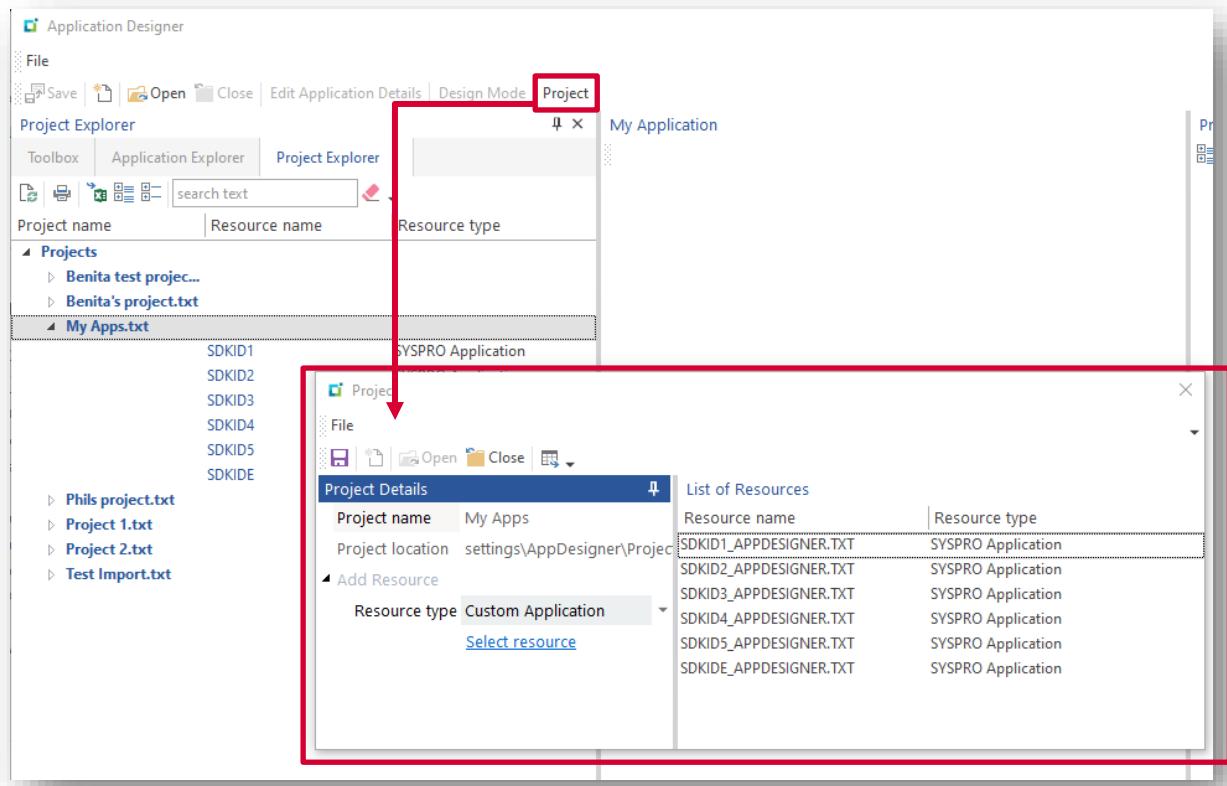
Projects can contain resources including:

- Custom Applications
- Standard SYSPRO applications
- Web Views
- .NET User controls
- VBS Modules

Additionally, projects can include 'Other resources'. These can be any type of file in any folder location that can be added to the project. When importing such files, they will always be imported to the ...\\settings\\AppDesigner\\Resources folder on the target machine.

MANAGING PROJECTS

To manage Projects, click on the **Project** button in the Application Designer toolbar:



CREATING NEW PROJECTS

To create a new project, click on the **NEW** button and enter a project name.

- The name can be anything up to 40 characters long but cannot contain any special character, such as / ? : & \ * " < > | # or %.
- The project will be saved in the ...\\settings\\AppDesigner\\Projects folder.

You can add 'resources' to your project by first selecting the **Resource type** and then clicking **Select resource** hyperlink.

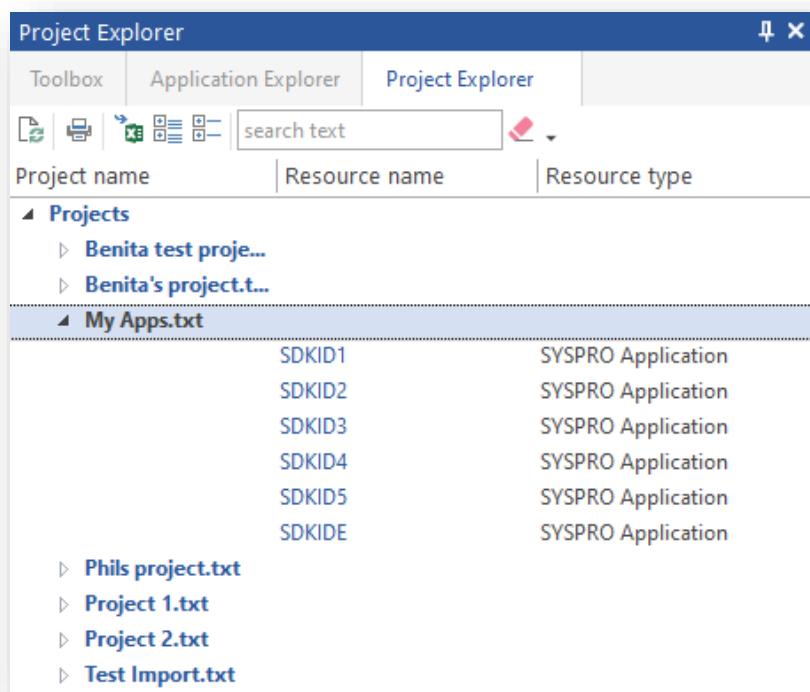
- The selected resources are shown in the **List of Resources** grid.
- You can delete a resource in the grid by pressing the DEL key.

You can also reposition any resource item by dragging it up or down the grid.

- You might wish to do this as when the project is exported and then imported on another machine, the list will be shown exactly in the same order.
- You might wish to add a 'other resource' to your project which contains installation or deployment instructions to the end-user, and you want that resource to appear at the top of the list.

After clicking **SAVE**, the resource list is saved in the projects named file. You can view this list either within the Projects window by clicking the **OPEN** button and selecting the appropriate project, or by using the Project Explorer.

The Project Explorer displays all the projects on the local machine:



MAINTAINING PROJECTS

You can click on the Project name to edit the project directly, or you can click on any application to edit it directly. This makes it easy to manage a group of applications, both from an editing perspective and from an export perspective.

EXPORTING A PROJECT

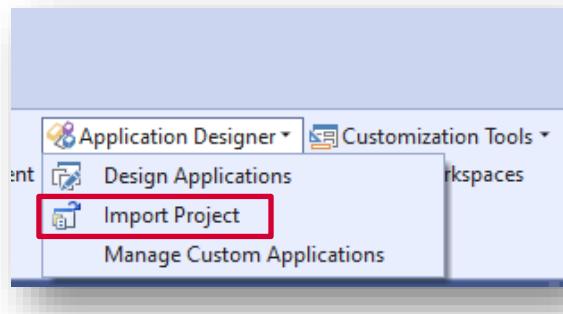
When editing a Project, you can click on the **Export** button.

- This will ZIP up the resource files and place the ZIPPED file in your ...\\AppDesigner\\Projects folder.
- The name of this zip file will be the name of the project with the suffix .prjzip.

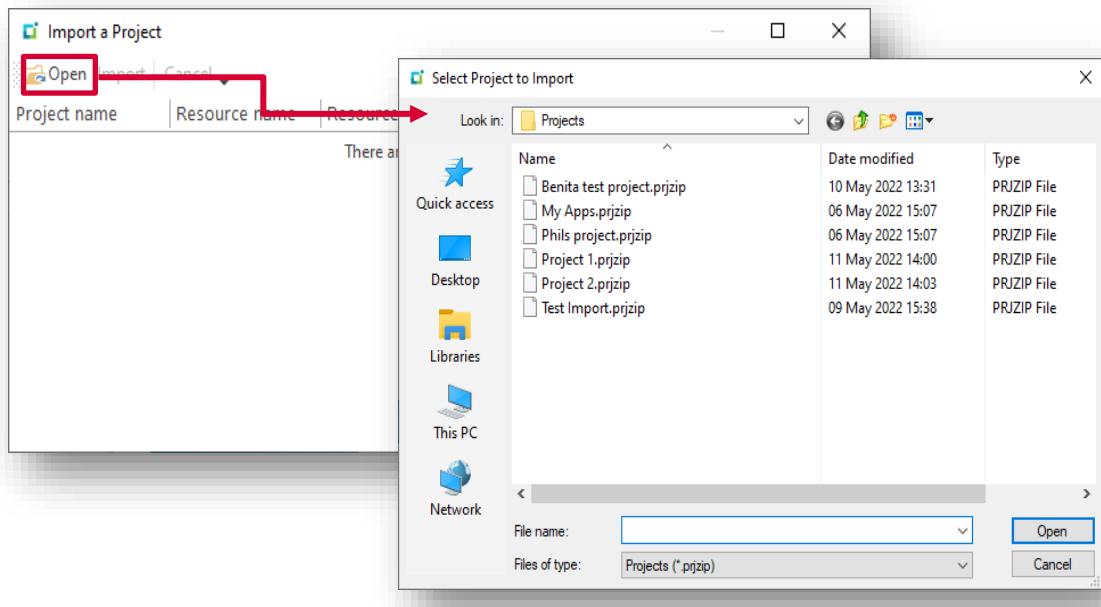
The ZIP file can now be sent (via email, for example) to the end-user.

IMPORTING A PROJECT

An end-user can import a project by clicking on the **Import Project** button in the Administration tab of the ribbon bar:



Once the Project Window is displayed, select the **Open** button to locate the ZIP file:



The contents of the project are shown in the Resource List:

Project name	Resource name	Resource type	Message
▲ My Apps			
SDKID1	SYS PRO Application	File is read-only	
SDKID2	SYS PRO Application	File is read-only	
SDKID3	SYS PRO Application	File is read-only	
SDKID4	SYS PRO Application	File exists	
SDKID5	SYS PRO Application	File exists	
SDKIDE	SYS PRO Application	File is read-only	

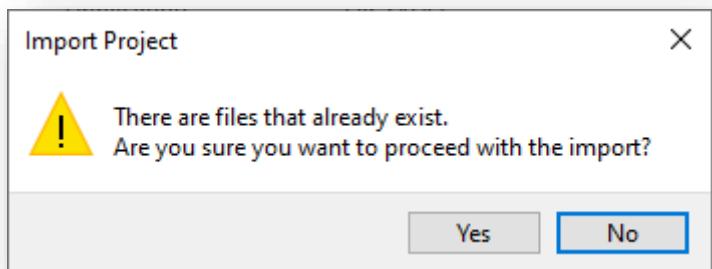
Select **Import** to import the resource contents.

- The files are unzipped from the ZIP file into a temporary folder in the ...\\Projects folder and are then copied to their respective locations on the client machine.
- If the system is running in client/server mode, the files are then copied back to the server machine, again to their respective folders.

Files imported into the ...\\AppDesigner\\Resources folder on the server are automatically self-healed to other client machines as users log in (in much the same manner as the contents of ManagedAssemblies and VBSModules folders).

Note: If any files are read-only, then the import will not be allowed to happen.

If any files already exist, the user will be prompted to decide whether to continue with the import:



USING THE {RESOURCE} KEYWORD

You can define a path using the keyword **{resources}** for these various controls:

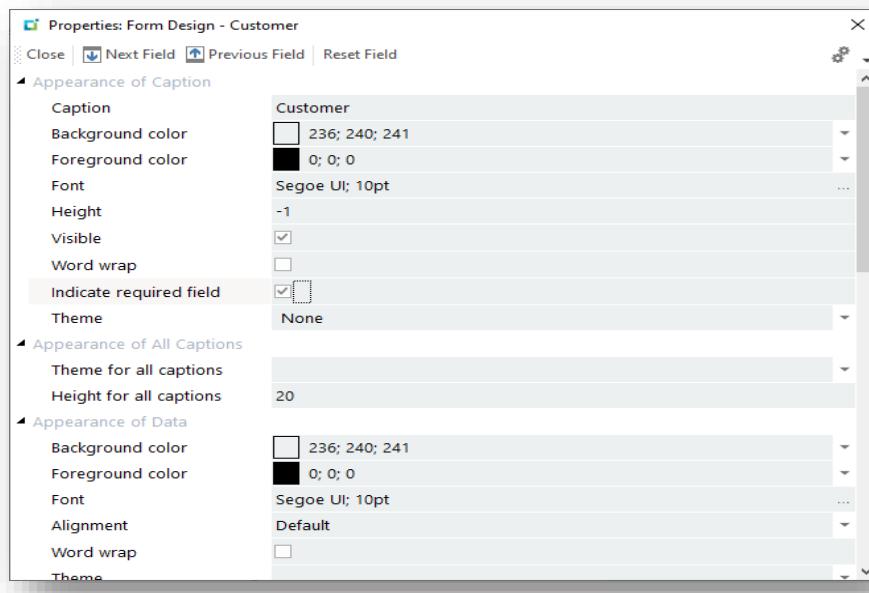
- Flowgraph
- Notepad
- Image viewer
- Document viewer
- Syntax text editor
- XAML control

These controls allow a specific file to be defined that will be loaded when the application runs. However, the entered path may not exist on another computer and therefore the application will fail to load the file.

To overcome this, you can enter a path starting with **{resources}**. This keyword will be changed at runtime (and when designing the application) to point to the ...settings\AppDesigner\Resources folder on both the local and target computer.

You would store your file(s) in this Resources folder and, using **Projects**, you can package up your solution including your resource files and send your solution as a ZIP file to the target machine for importing.

Here's an example of entering a path for the Flowgraph control:



Sample Apps

There are a series of sample Apps that ship with the product.

These Applications are intended to show different features of the designer and its controls and are fully working Applications.

The code is well documented and should be a good source of examples and techniques.

Sample Applications can be viewed when the **Open** button is pressed:

The screenshot shows a software interface titled "Open Application". At the top, there are buttons for "Select", "Copy", and "Print", followed by a search bar labeled "search text". Below the header is a table with columns: Application name, Application title, Date mod..., Read-only, Version, In development, and Category. The table is divided into sections: "Standard Applications", "Custom Applications" (which is highlighted with a blue background), and "Sample Applications". The "Sample Applications" section contains the following data:

Application name	Application title	Date mod...	Read-only	Version	In development	Category
SAMTAG	Sample Tag Cloud	06/03/24	<input checked="" type="checkbox"/>	6		
SAMPAG	Page Size Listview Sample	21/02/24	<input checked="" type="checkbox"/>	0		
SAMTR1	Customer Maintenance using Data Source	21/02/24	<input checked="" type="checkbox"/>	0		
SAMTVB	Bill of Materials Structure Sample	21/02/24	<input checked="" type="checkbox"/>	0		
SAMSRV	Web Service Example	20/02/24	<input checked="" type="checkbox"/>	2		
SAMCOC	Cocktail App	24/01/23	<input checked="" type="checkbox"/>	0	<input checked="" type="checkbox"/>	
SAMDTA	Data retrieval sample	24/01/23	<input checked="" type="checkbox"/>	0	<input checked="" type="checkbox"/>	
SAMTSK	Task Dialog Examples	24/01/23	<input checked="" type="checkbox"/>	2		
SAMWEA	Open Weather	24/01/23	<input checked="" type="checkbox"/>	14		
SAMXAM	XAML Application	24/01/23	<input checked="" type="checkbox"/>	0	<input checked="" type="checkbox"/>	
SAMXML	XML Queries Example	24/01/23	<input checked="" type="checkbox"/>	0	<input checked="" type="checkbox"/>	

Sample Apps are read only and cannot be amended. However, they can be copied by using the copy function at the top of the **Open Application** pane and renamed; thereafter they can be changed.

Appendix A – VBScript code snippets

This appendix contains some useful code snippets:

APP COMMENT SECTION

```
.....  
'App Name:  
'Date:  
'Authored by:  
.....  
'This application .....
```

CATCHING EVENTTYPE

```
'check for toolbar events  
if EventType = ToolbarEvent then  
    select case EventID  
        case Tbar_Close  
            call CallSYSPROFunction(CloseApp, " ", " ")  
  
        case Tbar_aaaaa  
            'do something here  
  
    end select  
  
end if
```

DISPLAYING ONUSEREVENT PARAMETERS

```
'comment out any line you do not want to display  
dim msg  
msg = ""  
msg = msg & "EventType: " & EventType & vbcrlf  
msg = msg & "EventName: " & EventName & vbcrlf  
msg = msg & "EventID: " & EventID & vbcrlf  
msg = msg & "Param1: " & param1 & vbcrlf  
msg = msg & "Param2: " & param2 & vbcrlf  
msg = msg & "Param3: " & param3 & vbcrlf  
  
msgbox msg
```

GET LOCAL WINDOWS TEMP DIRECTORY NAME

```
.....  
'Get local windows temp directory  
.....  
dim fso, tempdir  
set fso = CreateObject("Scripting.FileSystemObject")  
tempdir = fso.GetSpecialFolder(2)  'gets local temp directory
```

TOOLBAR CONSTANTS

```
'toolbar event ids to help make code more readable  
const Tbar_File = "01001"  
const Tbar_Close = "01002"  
const Tbar_aaaaa = "01003"
```

WRITING TEXT TO A FILE – SIMPLE FUNCTION

```
function WriteTextToFile(strtext, strfilename)  
dim fso  
dim f  
set fso = CreateObject("Scripting.FileSystemObject")  
set f = fso.CreateTextFile(strfilename,true)  
f.WriteLine(strtext)  
f.Close  
  
end function
```



www.syspro.com

Copyright © SYSPRO. All rights reserved.
All brand and product names are trademarks or
registered trademarks of their respective holders.

