Contents

Creating Web & Mobile Apps

Using Omnis JavaScript Client

Omnis Software Ltd

Released May 2023
Updated Jun 2023 Revision 35439
Updated Oct 2023 Revision 35659

About This Manual

This manual describes all the features in Omnis Studio that allow you to create applications that will run in a web browser on desktops and mobile devices, including tablets and phones. It describes how you create JavaScript Remote Forms, using the JavaScript Client and the JavaScript Components, for displaying your application in a browser or standalone app, as well as setting up the Omnis App Server for deploying your applications, either on your own server or in the cloud.

The information in this manual applies to all editions of Omnis Studio including the *Community Edition,* which allows you to create web and mobile applications using the Remote forms and JavaScript Client.

You will also need to consult parts of the Omnis Programming manual that describe Libraries and Classes, general Omnis programming techniques including SQL and List programming, as well as using the Studio Browser, Method Editor, Code Editor and the Omnis Debugger.

The *Omnis Reference* manuals contain information about all the Commands and Functions available in Omnis Studio, plus there is a comprehensive Help system, available from within the Omnis IDE using the F1 key, which contains a complete list of all Omnis Notation including all properties and methods.

If you are new to Omnis Studio

If you are new to Omnis Studio, you may like to work through the Tutorial which shows you how to connect to a **SQLite** database (provided in the download), and create a *JavaScript Remote Form* to browse the data in a web browser.

Alternatively, you may like to attend one of our free online training sessions in the **Omnis Academy** to help you to get to know Omnis Studio, including the *"Omnis Studio Basics"* course which introduces you to all the main tasks in building a web application in Omnis Studio; more information and registration is available on the Omnis website.

If you are evaluating Omnis Studio or wanting to prototype a web or mobile application quickly and easily, you may like to download the **Community Edition:** for more information and to register for the Community Edition, please go to our website.

When you start Omnis Studio

When you start Omnis Studio you will see the **Studio Browser;** if this is not visible, press the F2 key on Windows or Cmnd-2 on macOS. Under the **Hub** section you can look at example Omnis applications listed under the **Applets** and **Samples** options: you can open each example in your web browser or within Omnis itself, and you can examine the Omnis code in the associated library under the **Project Libraries** option in the Studio Browser (note you can open & run the sample apps in the Community Edition but access to the code is limited).

Copyright info

Chapter 1—Tutorial

The first section of this tutorial shows you how to create the *data classes* for an Omnis application to browse a picture database. The database contains sample client designs for a fictional design company including TV program identities, music album covers, and book jackets. You could, however, use the application to store any type of picture data, such as a library of your own favorite books or photos. To use the application for your own data, you need to work through the tutorial to create the application (library file) and then create your own database file.

Further sections of the tutorial show you how to create a *JavaScript Remote Form* so you can view the picture database in a web browser on your desktop, or on a mobile device, such as a phone or tablet.

What will you learn?

The tutorial shows you how to create the Omnis class structure required to match the data structure in an existing database (a SQLite database file available in the zip download), and how to create a *JavaScript Remote Form* to browse the data in a web browser, either on a desktop computer or a mobile device. The first part of the tutorial will take *about an hour to complete*, but you can pause at any stage and return to it at a later time, as long as you remember to open the database session using the SQL Browser each time you start Omnis Studio if you wish to browse through the data.

Download & install the Files

If you haven't already done so, you need to download the development version of Omnis Studio. For this tutorial, you can use the free 90-day trial version of the Professional edition, or the Community edition, which you can also download for free; both editions require registration.

Professional edition: www.omnis.net/developers/free-trial

Community edition: www.omnis.net/community-edition

Note that some sections of the tutorial cannot be completed using the Community Edition, since they relate to desktop classes only, but these can be skipped easily and you can create the web-based features without restriction.

To work through the tutorial, you will need to download and extract the project files from this **ZIP archive:** tutorial.zip. The archive contains a *SQLite* database file called **Pics.db** (the Pics.db-journal file is also required), and a *PNG* image file. Place the files from the zip in the **\welcome\tutorial** folder in the writable part of the Omnis tree you installed. Alternatively, you can place the tutorial files on your Desktop.

On Windows, you will find the 'tutorial' folder in the writable part of your Omnis installation, found in the 'AppData\Local' folder, for example:

C:\Users\<username>\AppData\Local\Omnis Software\ Omnis Studio <version>\welcome\tutorial

To find the 'AppData\Local' folder, you may need to enable hidden folders in the Windows File Manager and then navigate to the Omnis folder via Users\<username>\AppData\Local.

On macOS, you will find the Omnis files, including the 'tutorial' folder, in the Application Support folder, for example:

/Users/<username>/Library/Application Support/Omnis/Omnis Studio <version>/welcome/tutorial

To find the Application Support folder, click on the 'Go' menu in the macOS Finder, then *hold down the 'Option' key* and select the 'Library' option. There you will find the 'Application Support' folder and within that the 'Omnis' folder.

Tutorial Libraries

If you are using the Professional Edition (or the free trial), you can open and examine the libraries in the **final** folder in the '\welcome\tutorial\' folder in the Omnis folder; note you cannot open these libraries in the Community Edition.

The **PICS.LBS** library in the **final** folder contains the same classes as covered in this tutorial. The **PICS2.LBS** library has the same classes *with some extra classes,* including a Query class, a Report class, and a Menu class that enhance the Desktop aspects of the application. You can open these libraries and examine the code in both, or you can copy code from these into your tutorial library to save time when completing the different exercises.

Mouse and Keyboard Usage

In this tutorial, all mouse and key combinations are given in shorthand for all supported platforms, with the Windows shortcut first. So **"Press Ctrl/Cmnd-T"** means you press **Ctrl-T** under Windows, or press **Cmnd-T** on macOS. Similarly, **"Press F2/Cmnd-2"** means you press the **F2** key under Windows, or press **Cmnd-2** on macOS, or if your keyboard has function keys you can use **Fn+F2** on macOS.

Context Menus

Many of the design tools and class editors in Omnis have **Context Menus** that provide useful options that speed up development and navigation. To show a context menu, when using a *two-button mouse or trackpad*, click the **Right** button on the window or editor and select an option, or when using *a single button pointer*, hold down the **Ctrl** button and click the pointer; both these methods will be described in the tutorial as a *"Right-click"*.

Starting Omnis

To create an Omnis Studio application, you need to use the **Development** version. Once you have created your web or mobile app, you would run it in conjunction with the **App Server** version of Omnis Studio for which you need separate end-user client licenses (a free Web license is provided with the Community Edition). The Development version has all the design editors and debugging facilities that you need to help you build an application, but also allows you to test your application in a browser in "runtime mode" as you build it, without having to compile your app. During this tutorial, you will be using the Development version.

Studio Browser

When you start Omnis Studio you will see the **Studio Browser**, which is the main window from which you can access all the other tools and class editors in Omnis Studio. Initially you will see the **Project Libraries** option (highlighted below), where you will create a new Omnis library file. If you are new to Omnis or you are evaluating it, you may like to look in the **Hub** which contains many example libraries and sample code which you may find useful.

The above screen is the **Studio Browser** on macOS, but it will look identical on Windows, except for obvious differences in border or window style: since Omnis Studio is cross-platform, you should note that every part of the IDE performs in exactly the same way, with only minor differences in key strokes or menu/toolbar options.

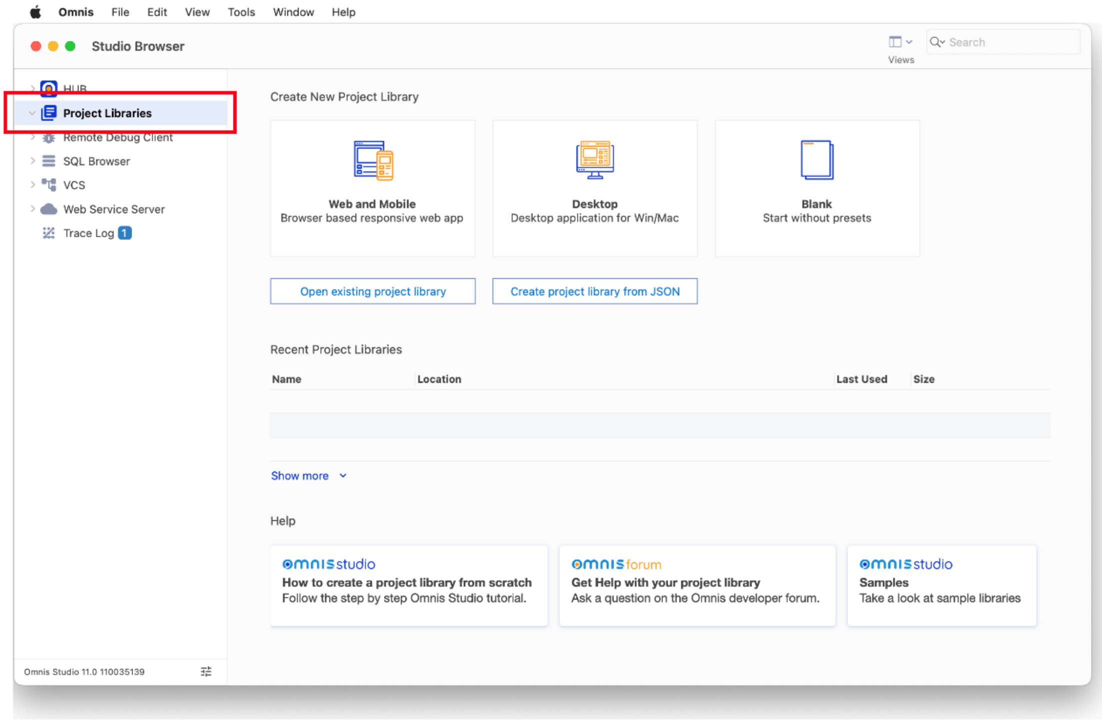

Omnis
File
Edit
View
Tools
Window
Help
Studio Browser
Views
Search
HUB
Project Libraries
Remote Debug Client
SQL Browser
VCS
Web Service Server
Trace Log 1
Omnis Studio 11.0 110035139
Create New Project Library
Web and Mobile
Browser based responsive web app
Desktop
Desktop application for Win/Mac
Blank
Start without presets
Open existing project library
Create project library from JSON
Recent Project Libraries
Name
Location
Last Used
Size
Show more
Help
omnis studio
How to create a project library from scratch
Follow the step by step Omnis Studio tutorial.
omnis forum
Get Help with your project library
Ask a question on the Omnis developer forum.
omnis studio
Samples
Take a look at sample libraries

Figure 1:

If you cannot see the Studio Browser press the **F2** key on Windows, or **Cmnd-2** on macOS. Under Windows, you can click on the **Browser** button (compass icon) on the main Omnis Studio toolbar, or you can select the **Browser** option from the **View** menu on the Omnis menubar. To show the main Omnis menubar on Windows you need to press the **Alt** key. On macOS, you can select the **Browser** option from the **View** menu on the Omnis menubar, or to show the Omnis toolbar, select the **Toolbars** option from View menu, show the View toolbar, click OK, and then you can click on the **Browser** button.

Creating a New Library

The starting point for your Omnis application or project is an **Omnis library.** A library stores all the *classes* in your application, including the web forms, data classes, and so on. If you want to edit an existing library, you can open it using the **Open existing project library** option in the Studio Browser.

You can import an Omnis library from a set of JSON files using the **Create project library from JSON** option, e.g. you can get sample Omnis libraries in JSON format from our GitHub repository at: https://github.com/OmnisStudio. However, for this tutorial, you will create a new blank library using the **Create New Project Library** section in the Studio Browser.

To create a new library:

- Press F2/Cmnd-2 or click on the **Studio Browser** to bring it to the top, then click on the **Project Libraries** branch in the Folders tree list; this should be selected by default when you first start Omnis
- Under the **Create New Project Library** section, click on the **Blank** option, as shown below (note the Desktop option is hidden in the Community edition)

The **Web and Mobile** option creates a new library containing a new Remote form and a Remote task, but in this tutorial we are going to build a library and Remote form from scratch or use the Class wizards to automate some parts of the process. If you are evaluating Omnis Studio, and haven't got time to work through the tutorial, you may like to try the **Web and Mobile** option and examine the Remote form and other classes it creates for you automatically.

- Having clicked the **Blank** option, navigate to the <Omnis Software>**\welcome\tutorial** folder in the writable part of your Omnis installation (or wherever you placed the tutorial files from the zip download).

On Windows, you may want to place the tutorial files in your 'AppData\Local' folder to allow read/write access to them (note that you may need to enable hidden folders in the Windows File Manager to access the AppData folder). On macOS you can save your library in the Documents folder or on the Desktop.

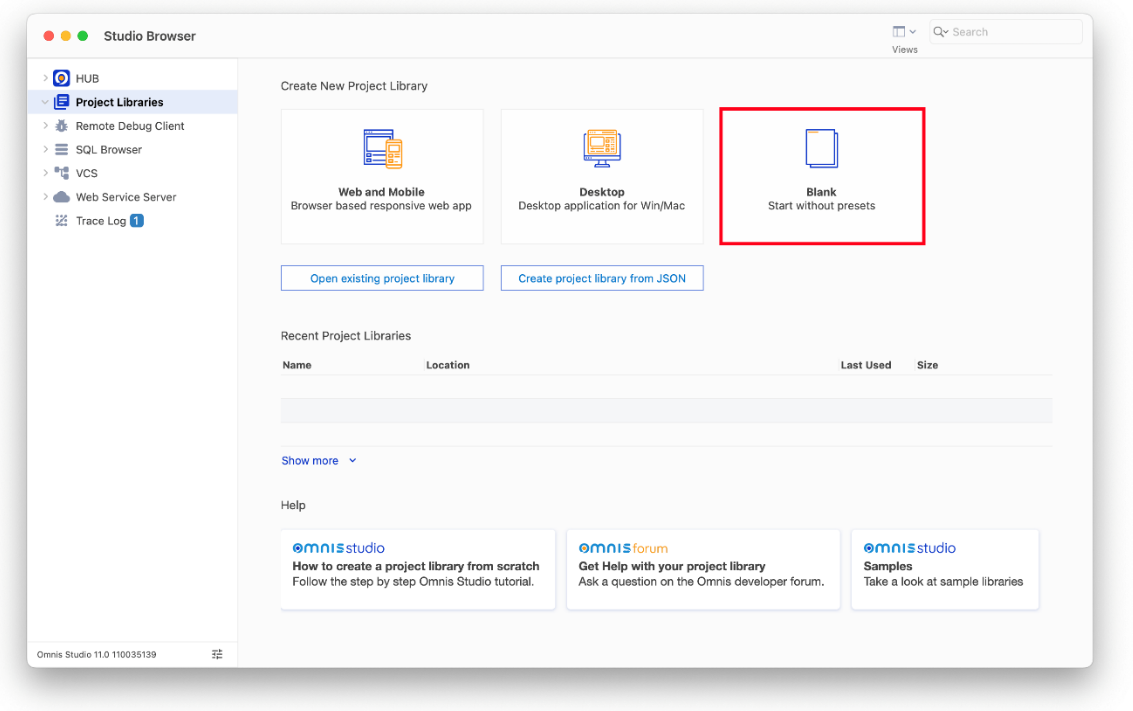
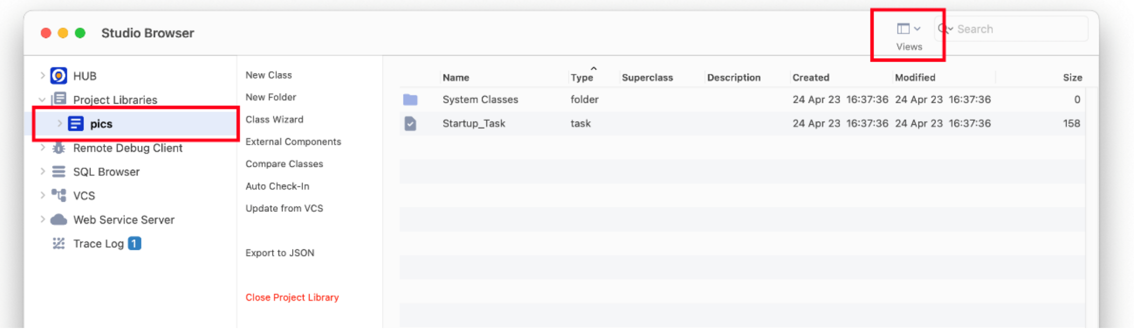

Studio Browser
Views
Search
HUB
Project Libraries
Remote Debug Client
SQL Browser
VCS
Web Service Server
Trace Log 1
Create New Project Library
Web and Mobile
Browser based responsive web app
Desktop
Desktop application for Win/Mac
Blank
Start without presets
Open existing project library
Create project library from JSON
Recent Project Libraries
Name
Location
Last Used
Size
Show more
Help
omnisstudio
How to create a project library from scratch
Follow the step by step Omnis Studio tutorial.
omnisforum
Get Help with your project library
Ask a question on the Omnis developer forum.
omnisstudio
Samples
Take a look at sample libraries
Omnis Studio 11.0 110035139

Figure 2:

- In the New Library dialog, type the name **pics.lbs,** including the .LBS extension, and click on **OK/Save.**

When you create or open a library it appears in the Studio Browser. To view the contents of a library you expand the 'Project Libraries' branch of the Folders tree list (on the left) and click on the library name, in this case the **pics** library; an alternative way to expand (or contract) a branch in the Folders tree list is to *double-click* on the name.



Name	Type	Superclass	Description	Created	Modified	Size
System Classes	folder			24 Apr 23 16:37:36	24 Apr 23 16:37:36	0
Startup_Task	task			24 Apr 23 16:37:36	24 Apr 23 16:37:36	158

Figure 3:

The classes that belong to the selected library are listed to the right of the Folders tree list in the Browser list. Using the **View** menu on the Browser window toolbar, you can change the display to Large Icons, Small Icons or Details (the default view). The following screenshot shows the Studio Browser in **Large Icons** view, which might be easier while you work through the Tutorial.

Note that your library initially contains a **Startup_Task** which is used to initialize the library when it starts up (i.e. when it is opened), as well as a folder containing some **System Classes** that are required to setup or configure a library. If you are using the Community Edition, your library will also include a **Remote_Task** class which will be listed in the Studio Browser.

Creating a Database Session

Before you start to build your application (project library), you need to open a session to the SQL database, to access the data you are going to use. In this case, you are going to connect your session to an existing SQLite database, provided in the download, that is already populated with data. However, you could use a database from another vendor, such as Oracle, Sybase, DB2, or MySQL, or some other data source connected via ODBC (your edition may restrict which database/s you can use). You can use the **SQL Browser**

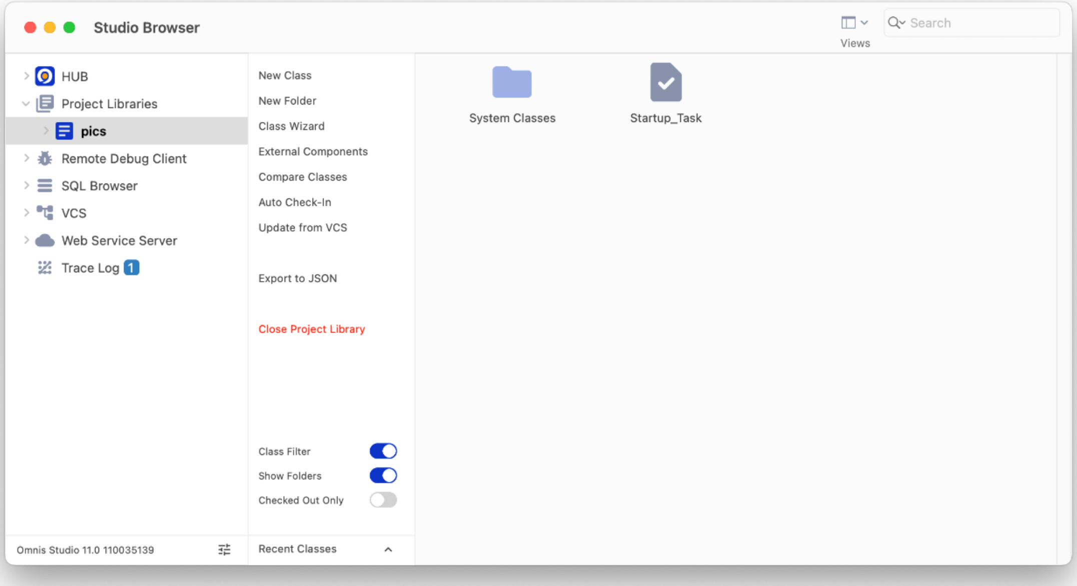


Figure 4:

during development to set up, modify and examine your database, or multiple databases on different servers, and you can use the SQL Browser to move database tables from one server or data source to another.

- Click the **SQL Browser** option in the Folders tree list in the Studio Browser and then select the **Session Manager** option to display all the defined session templates.

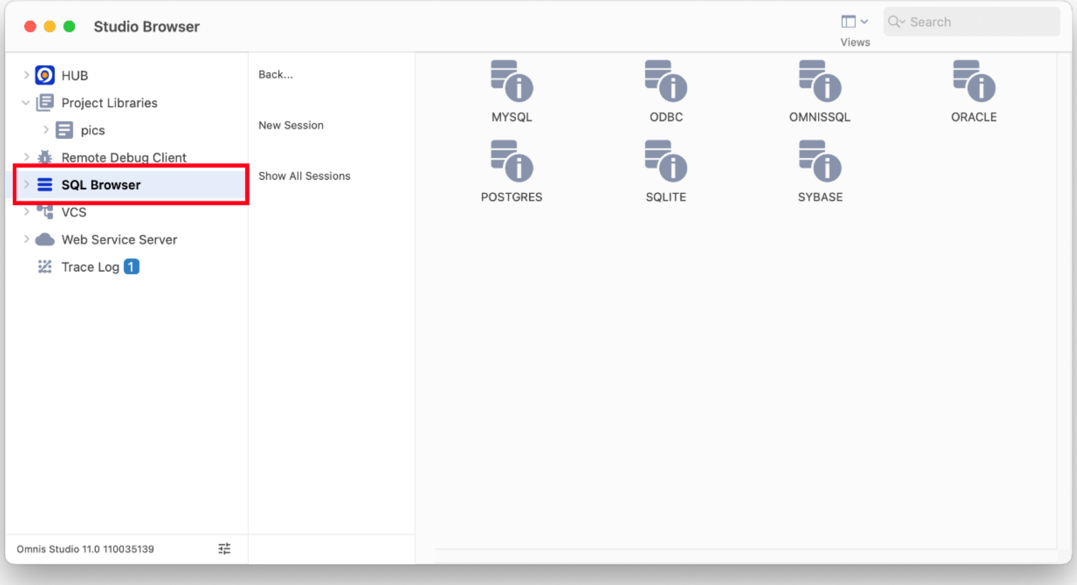


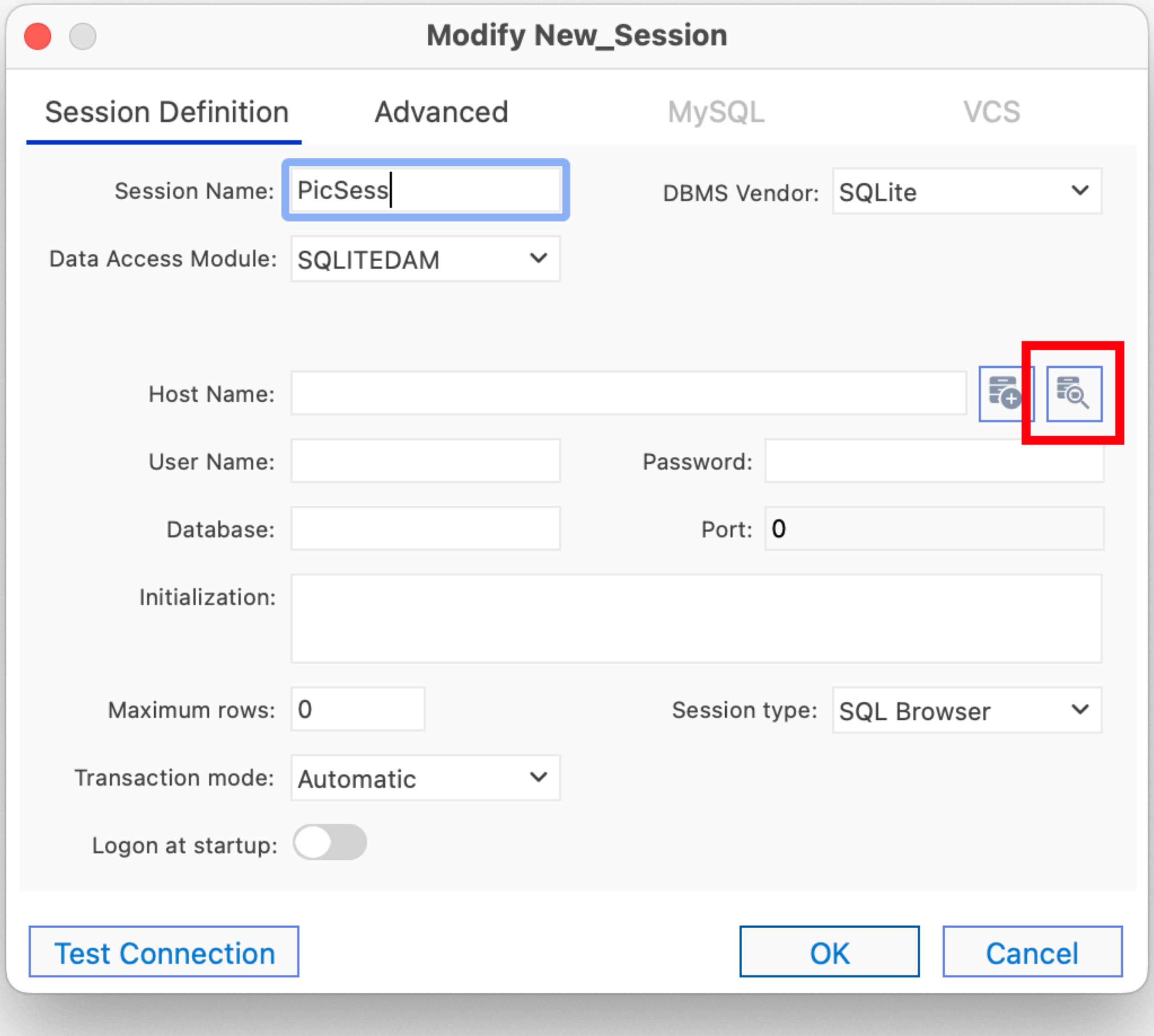
Figure 5:

Note that some of the session templates may not appear if your edition of Omnis Studio does not support a particular database, or you do not have the necessary clientware installed, but you should see templates for **PostgreSQL** and **SQLite** in all editions of Omnis Studio.

- Click the **New Session** option (in the options list in the middle) to create a new session template.

The **Modify Session** window will open which contains all the information needed to connect to your database. (Not all the fields on the session template are needed for all databases.)

- Type **PicSess** in the Session Name box, select **SQLite** from the DBMS Vendor list, select **SQLITEDAM** from the Data Access Module list; in this case, the DB Version is not required.



Session Name: PicSess

DBMS Vendor: SQLite

Data Access Module: SQLITEDAM

Host Name:

User Name:

Password:

Database:

Port: 0

Initialization:

Maximum rows: 0

Session type: SQL Browser

Transaction mode: Automatic

Logon at startup:

Figure 6:

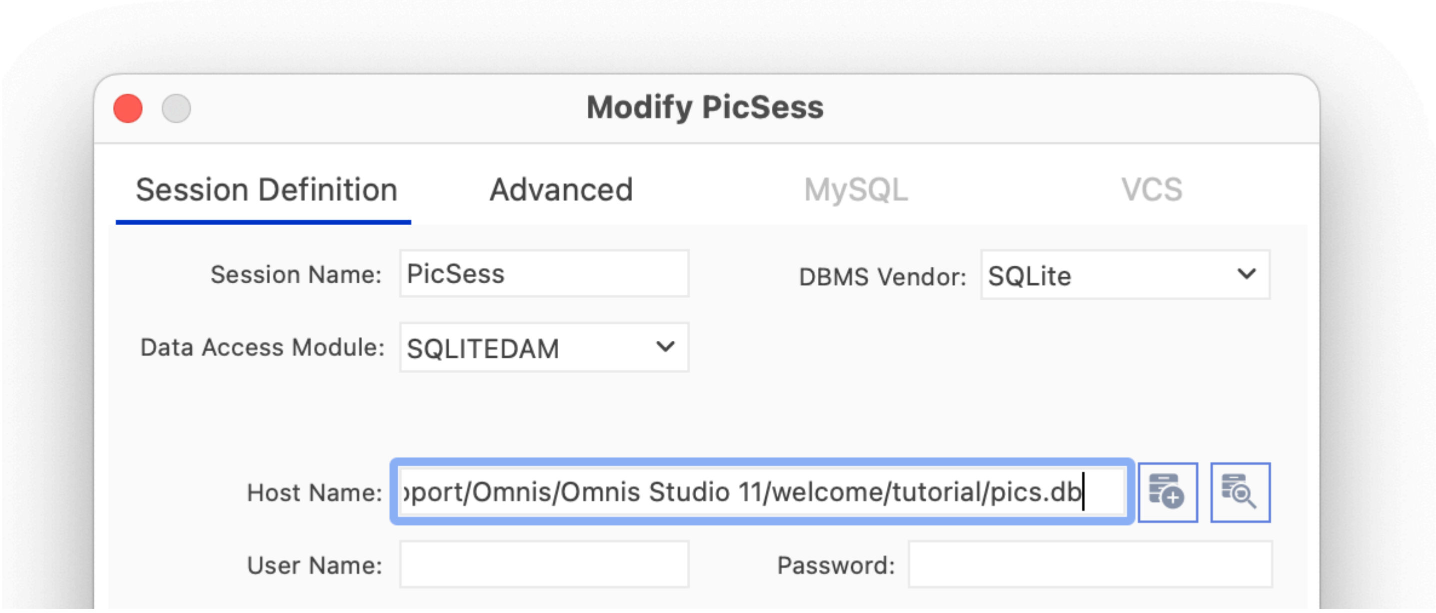
Now you need to connect this session template to the SQLite database available in the zip download. On the Modify Session window, the **Host Name** box needs to contain the path to the SQLite database file that you wish to use, in this case the Pics.db file you downloaded.

- Click the **Select Data File** button, located to the far right of the Host Name box (as shown highlighted above)
- Navigate to the 'pics.db' database file (provided in the zip download) located in the <Omnis Software>**\welcome\tutorial** folder in the writable part of your Omnis installation (or wherever you placed the file)

Under Windows, it will be located in AppData\Local such as:

C:\Users\<username>\AppData\Local\Omnis Software\Omnis Studio <version>\welcome\tutorial
On macOS, look for the 'tutorial' folder in the Application Support folder, such as:
/Users/<username>/Library/Application Support/Omnis/Omnis Studio <version>/welcome/tutorial

- When you have located the **pics.db** database file on your system, click OK and the path to this file will appear in the Host Name box.



Session Name: PicSess

DBMS Vendor: SQLite

Data Access Module: SQLITEDAM

Host Name: ›port/Omnis/Omnis Studio 11/welcome/tutorial/pics.db

User Name:

Password:

Figure 7:

Note that there is also a New Data File button on the Modify Session window that you can use to create a new empty SQLite database, but for this tutorial you will use the existing database file, as above.

- Now press the **OK** button to save the session template.

The new **PICSESS** session template will appear in the list of available session templates.

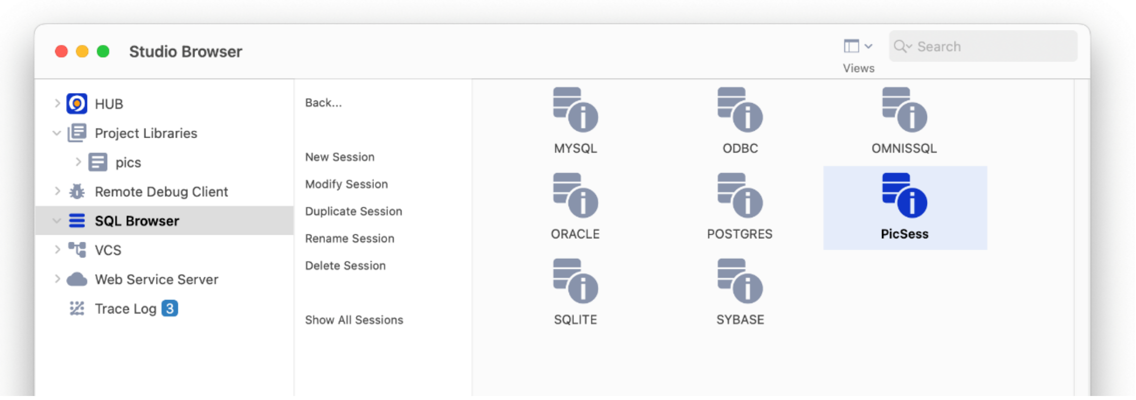


Figure 8:

1. Click on the **Back** option to exit the Session Manager

Opening a Database Session

To open the database session:

- Make sure the **SQL Browser** option is selected, then click on the **Open Session** option

You should see your PICSESS session template displayed in the list of sessions.

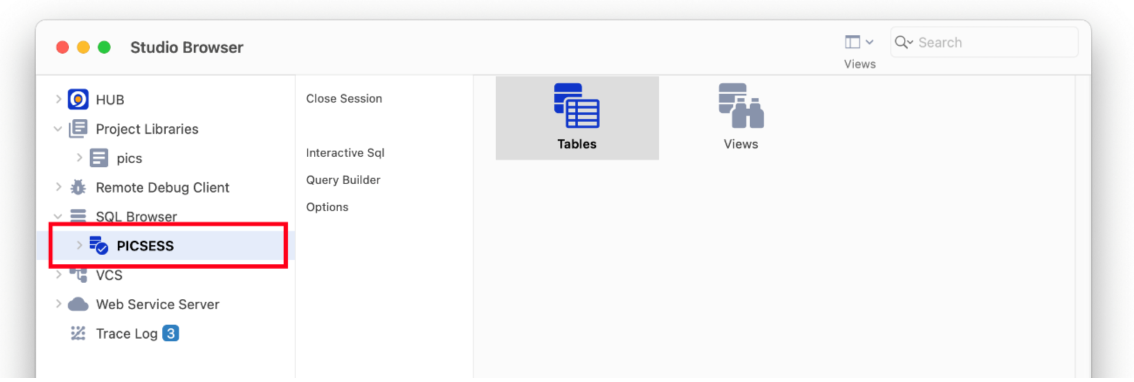


Figure 9:

- Select the **PICSESS** option to open the session, which is now shown highlighted.

The PICSESS session appears under the SQL Browser branch of the Folders tree list and the **Tables** and **Views** icons are displayed on the right. (Different databases may contain different database objects.)

Viewing your Tables

Note that if you have quit and restarted Omnis Studio, then you will need to reopen your database connection/session, as above, to continue this tutorial.

To view the tables contained in the PICS database:

- Expand the PICSESS branch of the Folders tree list and click on **Tables** in the tree list, or *double-click* on **Tables** icon in the Browser list; you will see the **MyPictures** table in the SQLite PICS database

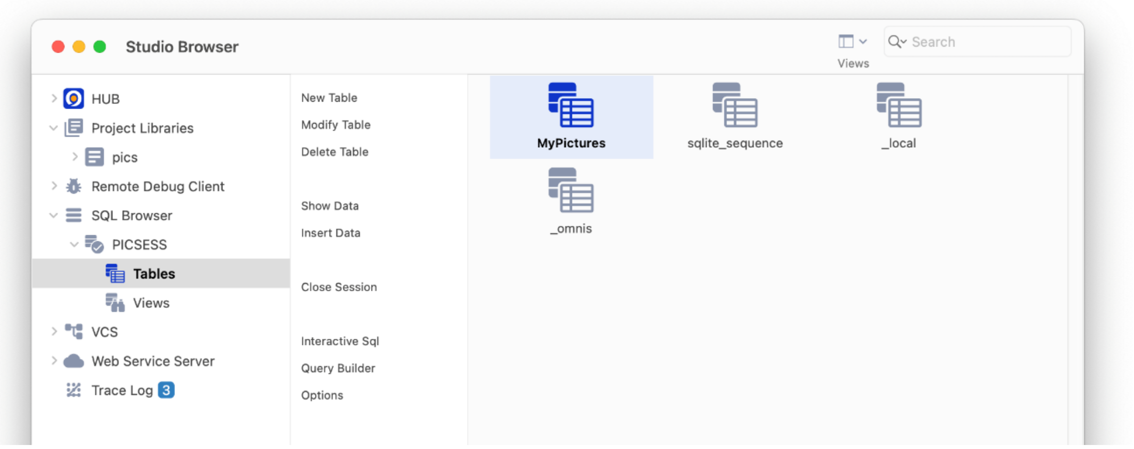


Figure 10:

- To view the column structure and indexes in the MyPictures table, *double-click* the **MyPictures** table in the Browser list (ignore the other system tables).

1. Close the Alter Table window by clicking on **Finished** or the close box.

Viewing your Data

To view the contents of the MyPictures table:

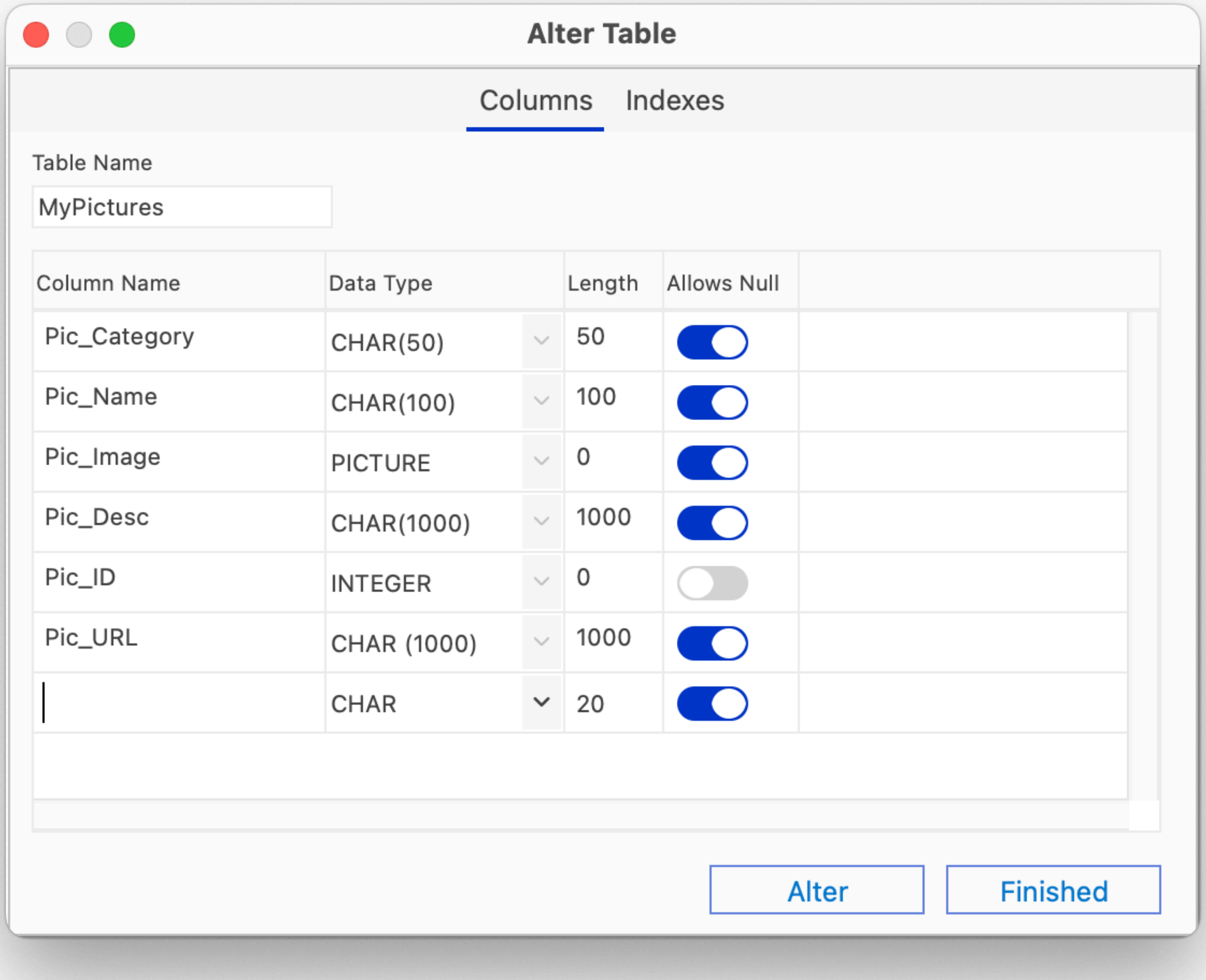


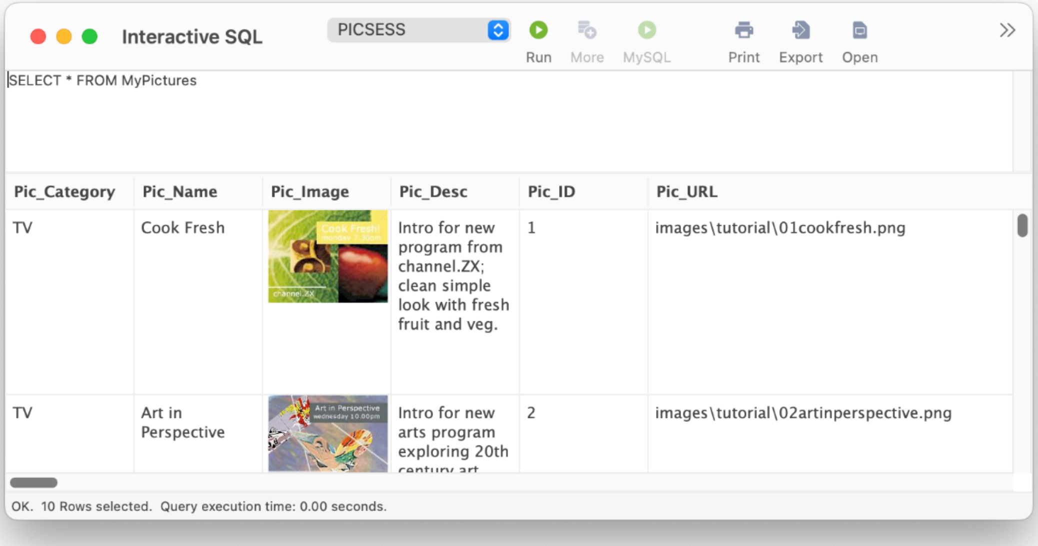
Table Name

MyPictures

Column Name	Data Type	Length	Allows Null
Pic_Category	CHAR(50)	50	●
Pic_Name	CHAR(100)	100	●
Pic_Image	PICTURE	0	●
Pic_Desc	CHAR(1000)	1000	●
Pic_ID	INTEGER	0	○
Pic_URL	CHAR (1000)	1000	●
	CHAR	20	●

Figure 11:

- Select the **MyPictures** table and click on the **Show Data** option, or right-click on the table and select **Show Data** from the context menu.
- Expand the **Interactive SQL** window, resize the Pic_Image column (by dragging the vertical line in the table header), and scroll down to view the data.



```
SELECT * FROM MyPictures
```

Pic_Category	Pic_Name	Pic_Image	Pic_Desc	Pic_ID	Pic_URL
TV	Cook Fresh		Intro for new program from channel.ZX; clean simple look with fresh fruit and veg.	1	images\tutorial\01cookfresh.png
TV	Art in Perspective		Intro for new arts program exploring 20th century art	2	images\tutorial\02artinperspective.png

OK. 10 Rows selected. Query execution time: 0.00 seconds.

Figure 12:

Note the **Interactive SQL** tool has entered the SQL needed to view the whole of the MyPictures table (SELECT * FROM MyPictures), but you can use this tool to return whatever data you like by entering your own SQL statements and clicking on the Run button. For example, you could add an **'Order By Pic_Category'** clause, then click Run to sort the data on Pic_Category; in this case the Books would appear at the top.

- When you have viewed the data, close the Interactive SQL window.

Now you have used the SQL Browser in the Development version of Omnis Studio to create the PicSess session, and to explore the tables and data in the database, you need to provide a way for your library to access the data in the database, using that session information.

Making a Schema

Note that if you have quit and restarted Omnis Studio, then you will need to reopen your database connection/session, as described in the 'Opening a Database Session' section, to continue this tutorial.

To allow your library to access the data in the MyPictures table you must first create a **Schema class** in your library that defines the column structure of the table.

Make sure that the **Libraries** branch of the Folders tree list is expanded so that you can see the **PICS** library in the tree list (see below).

Then make sure that the **SQL Browser** branch of the Folders tree list is expanded so that you can see **Tables** under the PicSess session in the tree list.

- Click on the Tables branch of the Folders tree list to select it; you will see the table **MyPictures** listed in the Browser list.
- Drag the **MyPictures** table from the list of tables and drop it on the **PICS** library in the Folders tree list to create a schema class; drop the table when the PICS library is highlighted, as shown.

- Click on the **PICS** library in the Folders tree list.

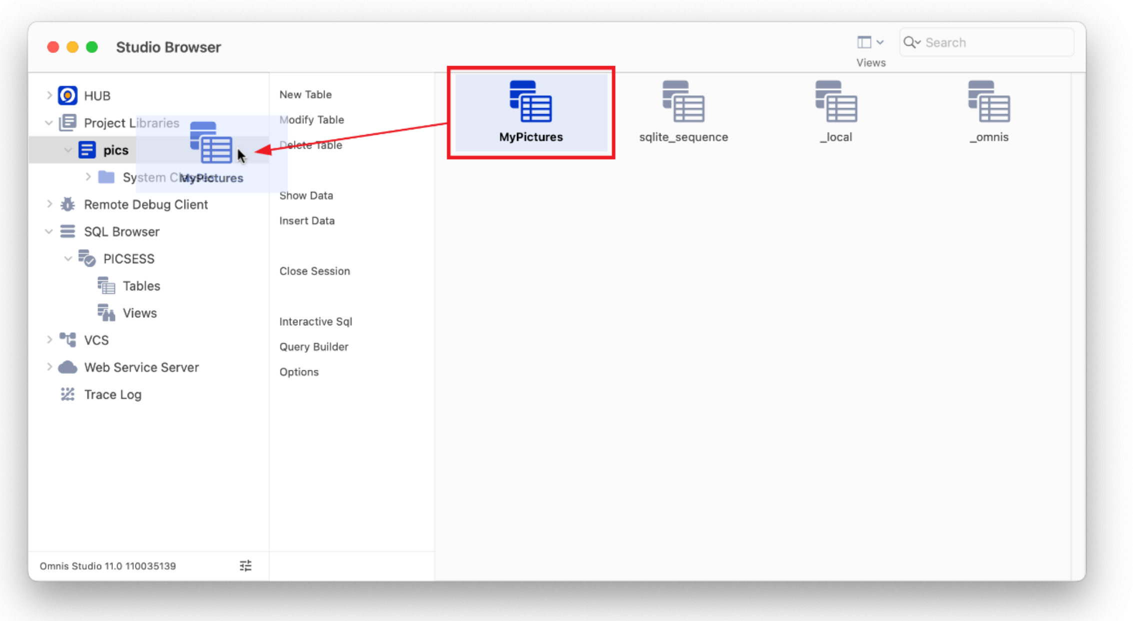


Figure 13:

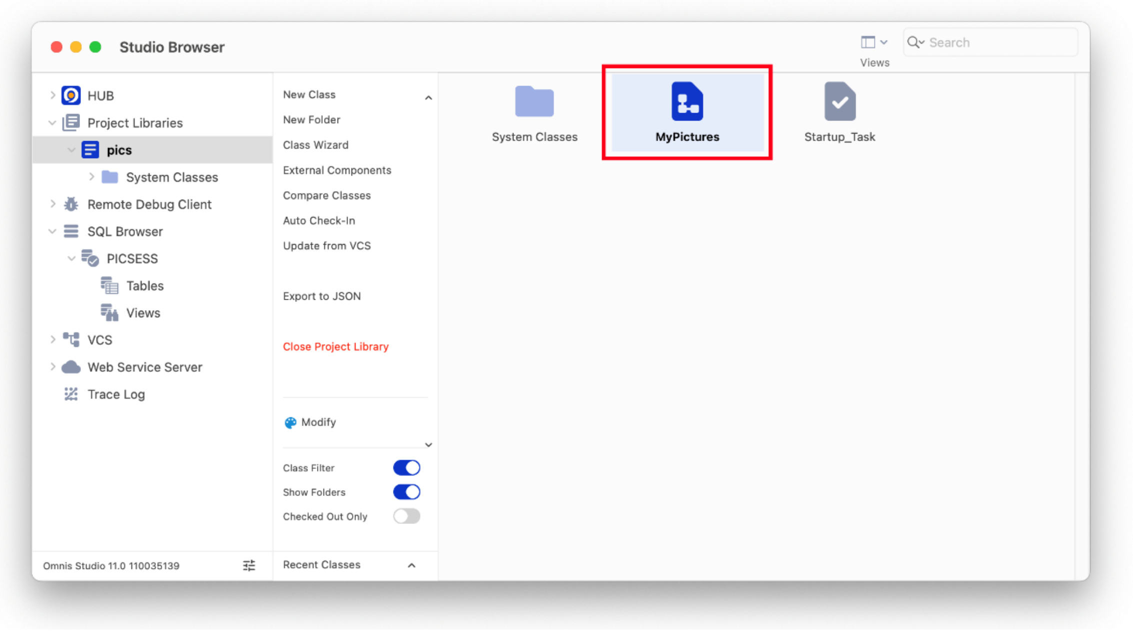


Figure 14:

You should see a new schema class called **MyPictures** appear in the Browser list.

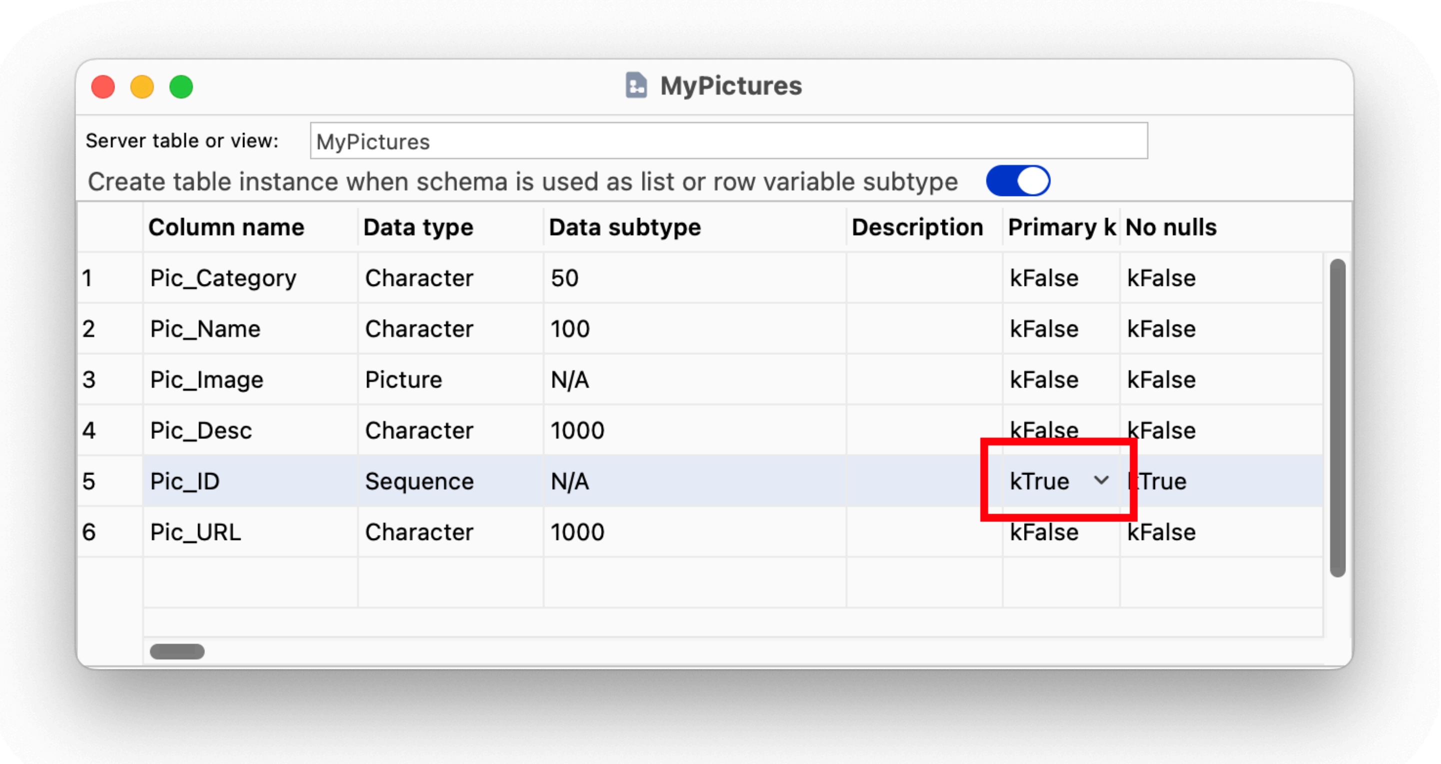
This schema class was created by Omnis automatically to match the definition of the database table when you dropped the MyPictures table onto the PICS library. (If you create your own app using your own database, you could do the same using the tables in your database, that is, drag a table onto your library to create a schema class.)

Editing a Schema

To edit the schema class:

- Double-click the **MyPictures** schema class in the Browser list to open the schema editor. Alternatively, click the **MyPictures** schema class and then select the **Modify** option.

The 'Server table or view' box at the top of the schema editor contains the name of the database table **MyPictures** that is related to this Omnis schema class. Each of the entries in the schema contains the name and Omnis data type of one of the columns in the table. Note the data type is the equivalent Omnis data type to that of the column defined in the SQLite database table.



Server table or view: MyPictures

Create table instance when schema is used as list or row variable subtype

	Column name	Data type	Data subtype	Description	Primary k	No nulls
1	Pic_Category	Character	50		kFalse	kFalse
2	Pic_Name	Character	100		kFalse	kFalse
3	Pic_Image	Picture	N/A		kFalse	kFalse
4	Pic_Desc	Character	1000		kFalse	kFalse
5	Pic_ID	Sequence	N/A		kTrue	kTrue
6	Pic_URL	Character	1000		kFalse	kFalse

Figure 15:

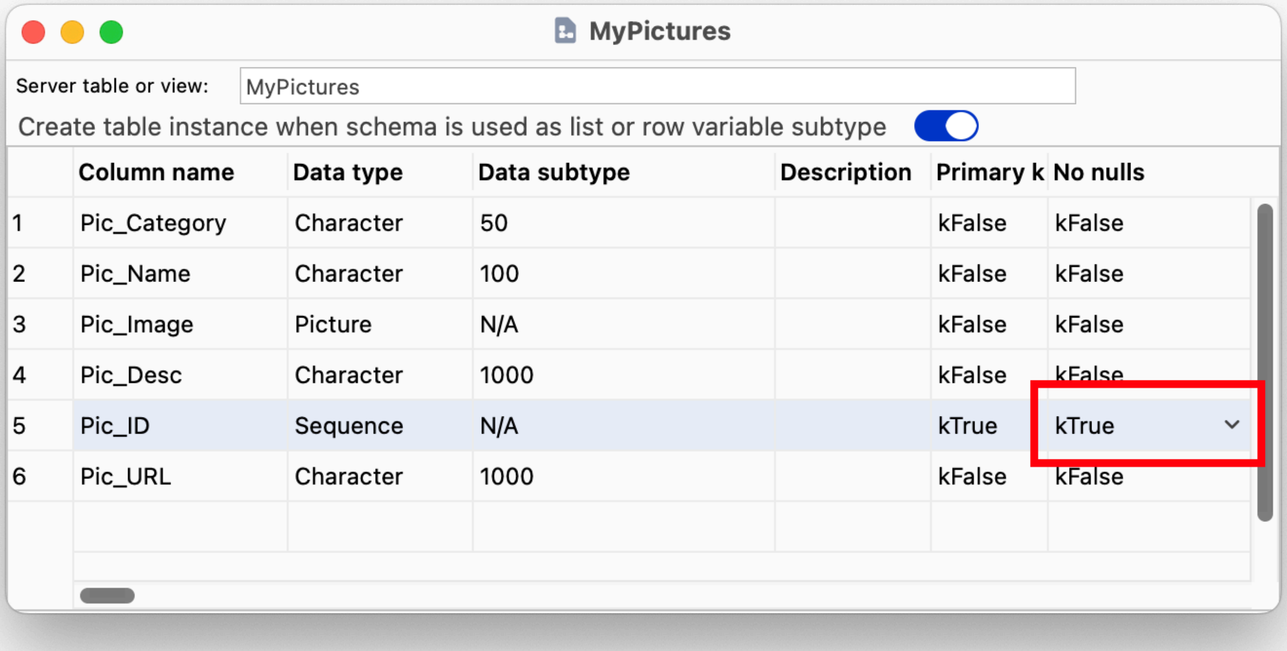
To ensure that certain kinds of queries execute in the most efficient manner it is recommended that one column in each table should be designated as the *Primary Key*. It is important that the value of the primary key column in each row is unique because it is used to identify a particular row of data when it is to be updated or deleted. In other words, only one row in each table should contain a specific value in the primary key column.

- Locate the **Pic_ID** row (it should be row 5, as above) in the schema class and make sure the value of the **Primary Key** property is set to **kTrue** (As you use Omnis Studio, you will meet many constants such as kTrue, all of which by convention start with the letter "k"). Omnis should have done this automatically since it tries to identify the primary key

In addition, Omnis sets the **No nulls** column to kTrue, so the value is incremented automatically every time a new record is entered.

When you have finished modifying a class in Omnis you can simply close the editor window to save it, or you can use the **Save** option from the *File menu*, or press **Ctrl/Cmnd-S.** You can also select the **Auto Save** option from the File menu to force Omnis to save any updates automatically.

1. Close the MyPictures schema class to save it.



MyPictures

Server table or view: MyPictures

Create table instance when schema is used as list or row variable subtype ●

	Column name	Data type	Data subtype	Description	Primary k	No nulls
1	Pic_Category	Character	50		kFalse	kFalse
2	Pic_Name	Character	100		kFalse	kFalse
3	Pic_Image	Picture	N/A		kFalse	kFalse
4	Pic_Desc	Character	1000		kFalse	kFalse
5	Pic_ID	Sequence	N/A		kTrue	kTrue
6	Pic_URL	Character	1000		kFalse	kFalse

Figure 16:

Creating a Desktop Form using a wizard

IMPORTANT: the following section shows how you can create a desktop form, so if you are only interested in creating a web or mobile app (or you are using the Community Edition), you can skip this section and jump to the section 'Creating a Web Form from scratch', or the 'Creating a Web Form using a wizard' section.

Having created the data structure for your database (the MyPictures schema), you can create a window or form to browse and insert data locally using a *window class*.

Remember, if you have quit and restarted Omnis Studio, then you will need to reopen your database session, as described in the 'Opening a Database Session' section.

- Click the **Pics** library in the Folders tree list in the Studio Browser and select the **Class Wizard** option on the left of the Browser list to display all the available wizard types.

- Now click on the **Window…** option on the left of the Browser list to display the available window class wizards.

- Click the **SQL Form Wizard** in the Browser list.

- At the bottom of the Browser window type the class name **PicsWindow** and press the **Create** button.

The radio buttons on the Omnis Class Wizard allow you to select the style of form to create. Select the first radio button that reads *"One field per column based on schema or query class".*

Note that if different text is displayed next to the first radio button, you have selected the wrong type of wizard so you should press the **Cancel** button on the wizard window and choose the **SQL Form Wizard.**

- Press the **Next** button to move on to the stage where you will choose the data structure to be used to create the form.

Note that if at this point you get the message *"There are no file classes available for selection. Please create a file class in your library and try again"*, you have selected the wrong type of wizard so you should press the **OK** button and choose the SQL Form Wizard.

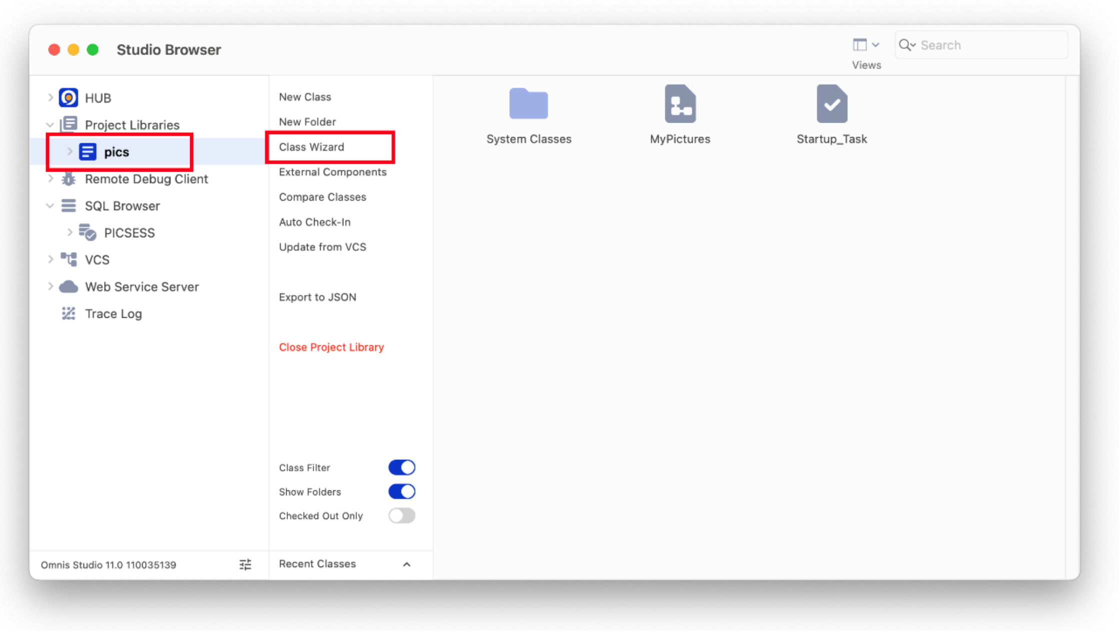


Figure 17:

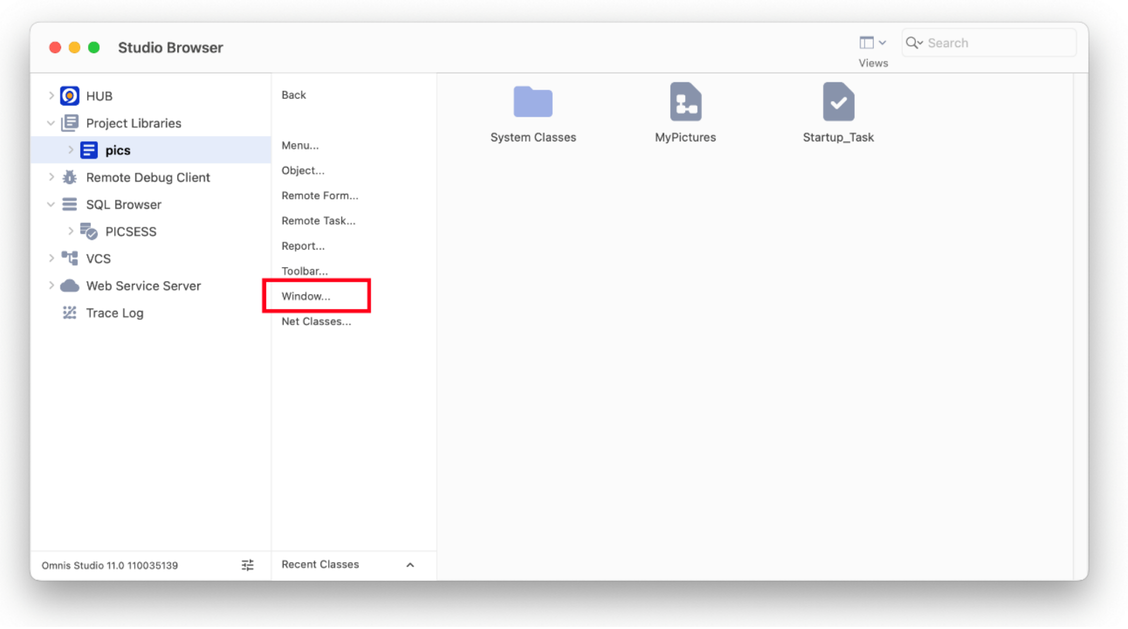
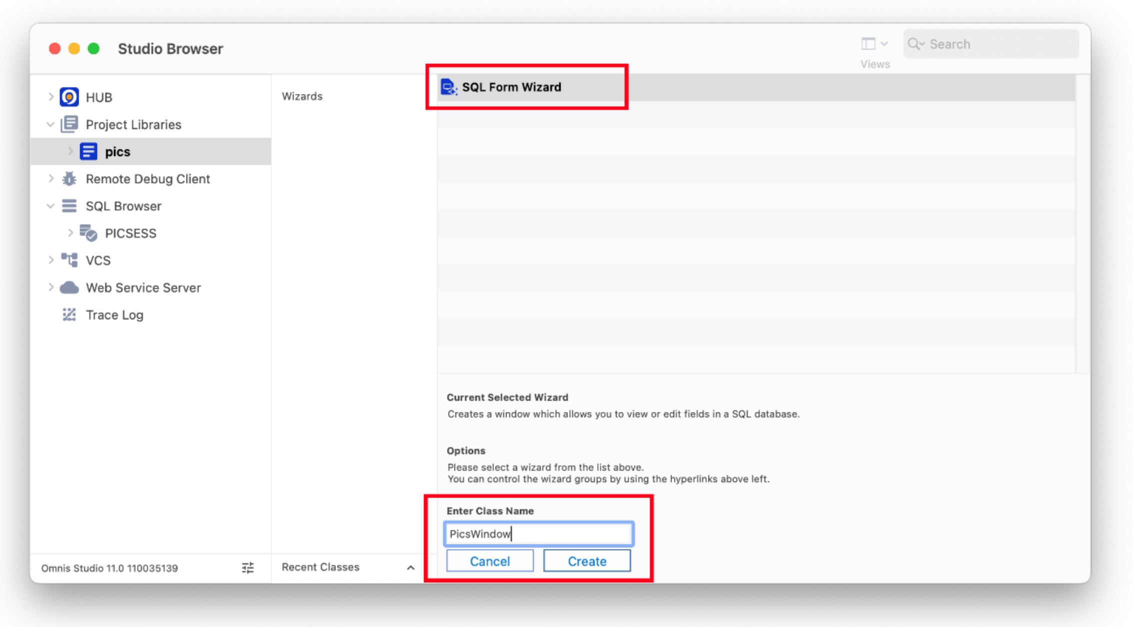


Figure 18:



Current Selected Wizard
Creates a window which allows you to view or edit fields in a SQL database.

Options
Please select a wizard from the list above.
You can control the wizard groups by using the hyperlinks above left.

Enter Class Name: PicsWindow

Figure 19:

- Click the check box to select the **MyPictures** schema and press the **Next** button to move on to the stage where you will choose the database connection that will be used to access the data.

Note that if at this point you get the message *"There are no SQL sessions available for selection. Please use the SQL Object Browser to open a session and try again"*, you need to cancel the wizard and open a database connection, as described earlier.

- Enable the switch to select the **PICSESS** database session and press the **Next** button to move on to the final stage.
- Press the **Finish** button to create the form.

Note that at any time prior to pressing the Finish button you can press the **Previous** button to go back and review or change a selection.

Editing a Desktop Form

After the wizard has finished creating the form, the Window Class Editor will open with the new **PicsWindow** window class in design mode, ready for you to modify as you wish. The *Component Store* (on the left-hand side) and *Property Manager* (on the right) are opened automatically.

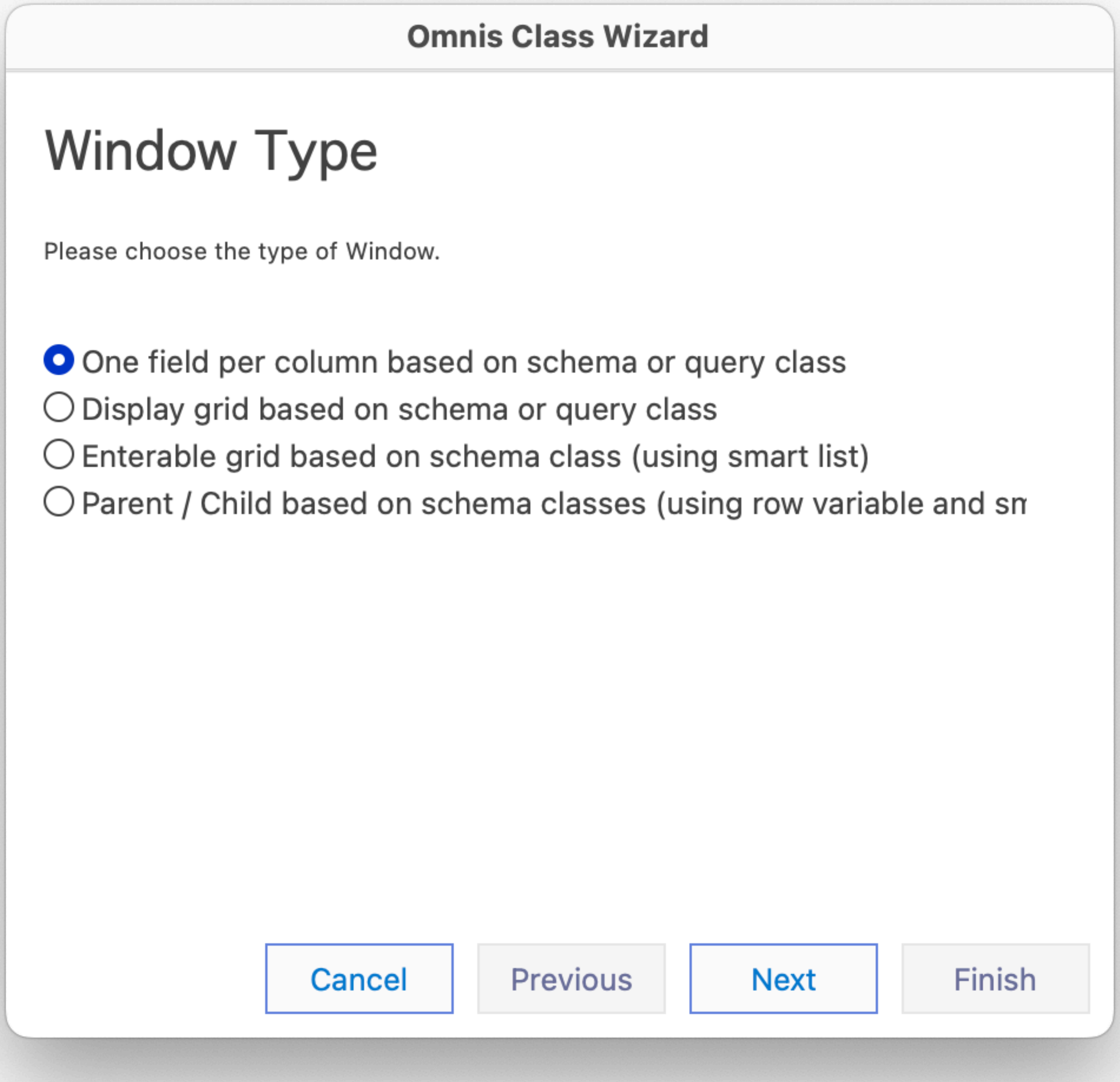
- Resize the window (and/or reposition/resize any fields if necessary) so that you can see all of the fields, for example, you can widen the image field (Pic_Image).

The **Component Store** (left) is a palette containing the objects and fields that you can add to the forms and windows in your library, while the **Property Manager** (right) lets you view and modify the values of properties of the currently selected object, e.g. in this case, the window class. At this stage, you do not need to use these tools.

Adding some code

Before you can open the window and use it to insert data, you must make a small change to the Insert button. Doing this will introduce you to the Omnis *Method Editor* which is an all-purpose tool with built-in debugger for adding Omnis code to the objects and classes in your library.

The **Pic_ID** column in the SQLite database we are using is an INTEGER data type with a *Primary Key*, known as an INTEGER PRIMARY KEY, which will store a unique integer value automatically. The "trick" is to insert the value of **Pic_ID** as **NULL** and SQLite will insert the next available numeric value; in effect the value of Pic_ID is incremented by 1 automatically and the new value is inserted. If you delete a data record the value of Pic_ID in that record is never reused; SQLite will always use a new unique value when inserting a new row.



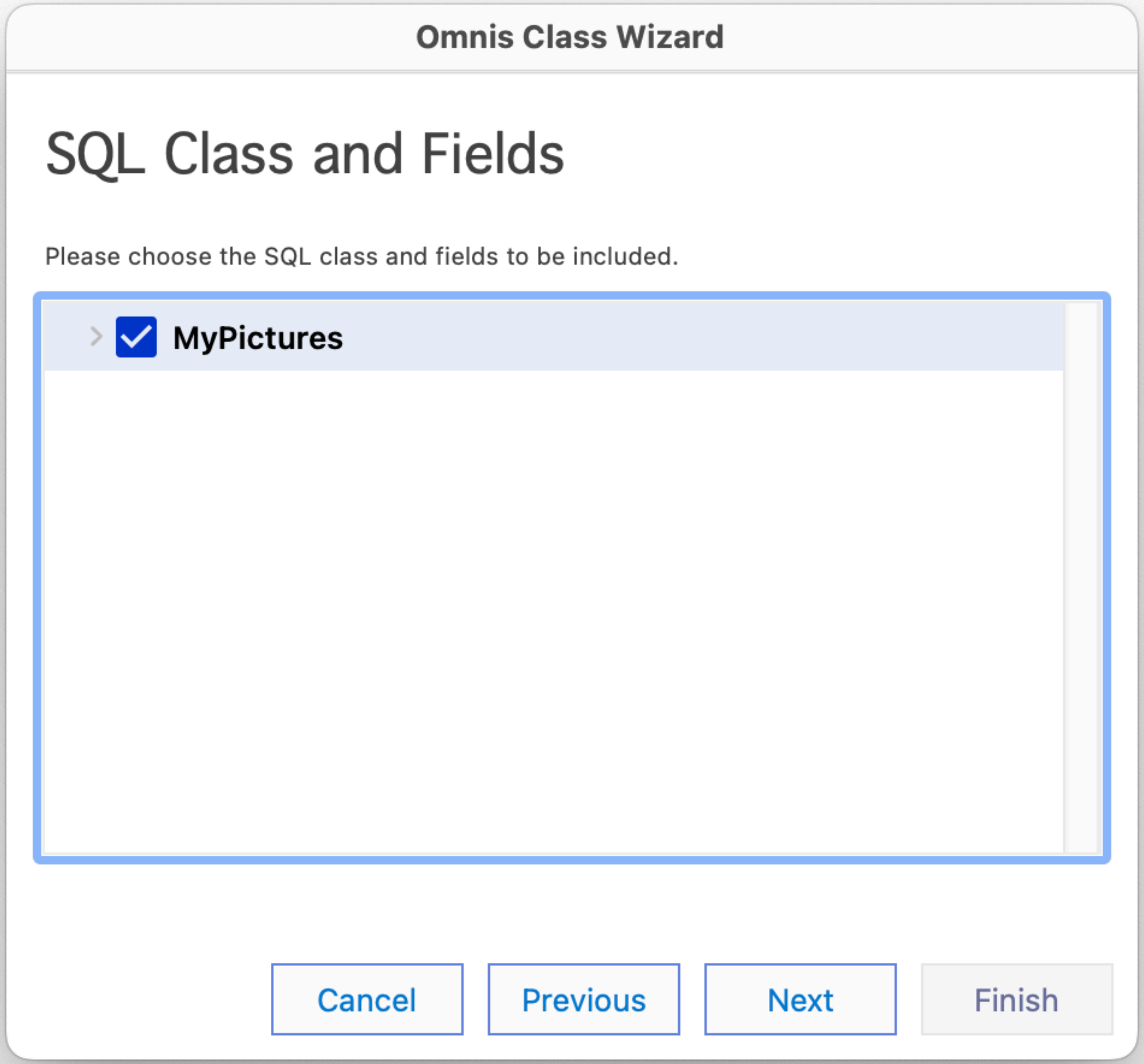
Omnis Class Wizard

Window Type

Please choose the type of Window.

● One field per column based on schema or query class
○ Display grid based on schema or query class
○ Enterable grid based on schema class (using smart list)
○ Parent / Child based on schema classes (using row variable and sn

Figure 20:



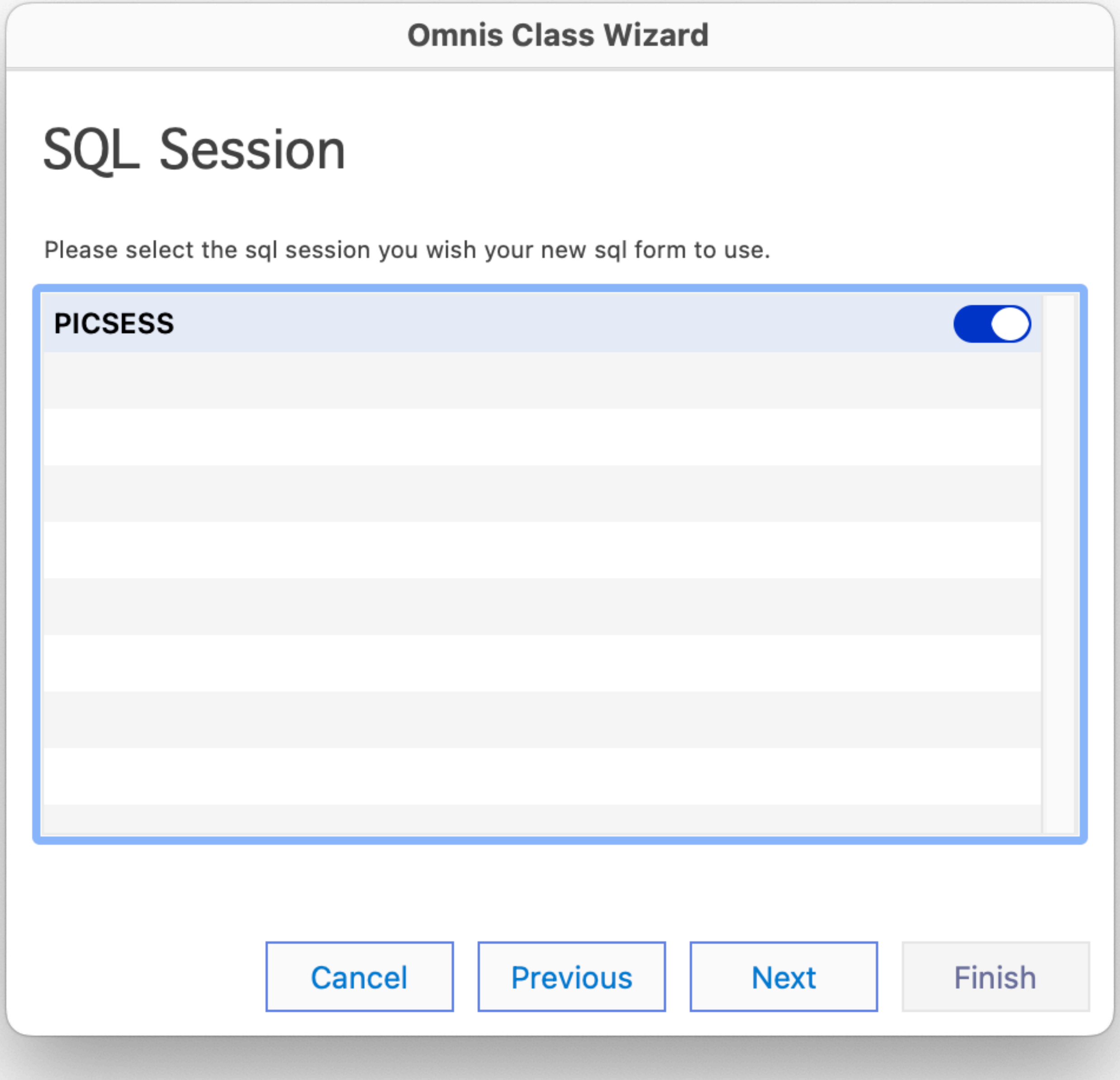
Omnis Class Wizard

SQL Class and Fields

Please choose the SQL class and fields to be included.

☑ MyPictures

Figure 21:



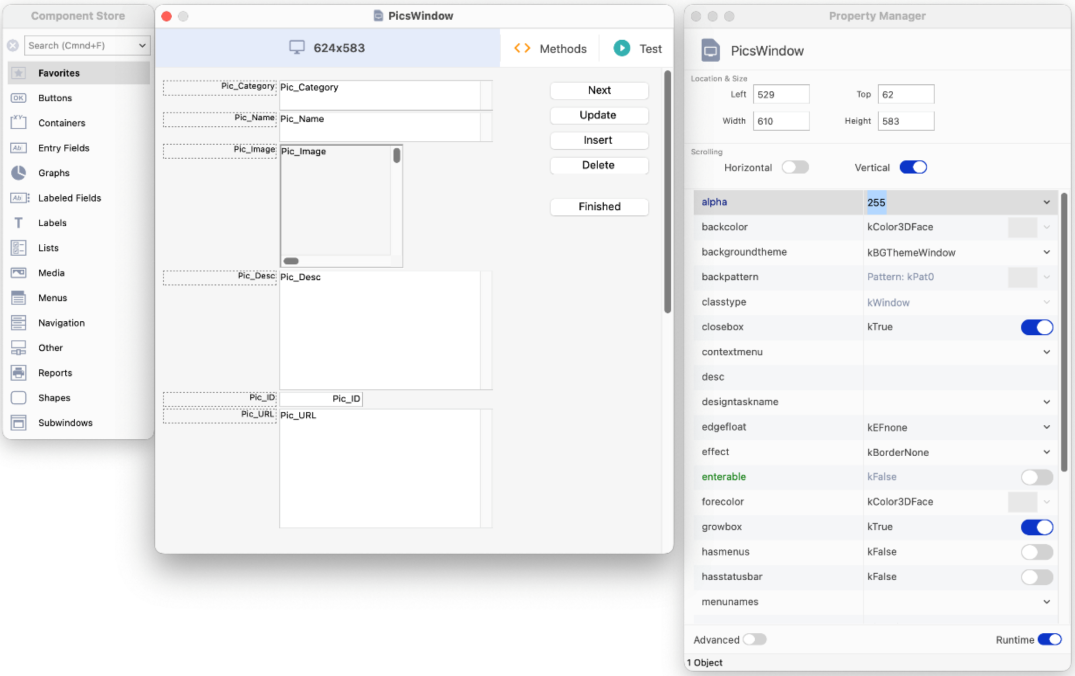
Omnis Class Wizard

SQL Session

Please select the sql session you wish your new sql form to use.

PICSESS

Figure 22:



Location & Size

Left	529	Top	62
Width	610	Height	583

Scrolling

Horizontal ○ Vertical ●

alpha	255
backcolor	kColor3DFace
backgroundtheme	kBGThemeWindow
backpattern	Pattern: kPat0
classtype	kWindow
closebox	kTrue
contextmenu	
desc	
designtaskname	
edgefloat	kEFnone
effect	kBorderNone
enterable	kFalse
forecolor	kColor3DFace
growbox	kTrue
hasmenus	kFalse
hasstatusbar	kFalse
menunames	

Figure 23:

- Assuming you have the **PicsWindow** open in design mode, ***double-click*** on the **Insert** button, or Right-click on the button and select **Field Methods…** (or select the button and press Shift-F8)

The Omnis Method Editor and Code Editor will open and the **$event** method behind the **Insert** button will be highlighted. This method will be executed when you click on the button; in this case the method inserts the current values in the iSqlRow variable into the database, and it's before this line you need to make a small addition.

- Click anywhere in the second line of code 'Do iSqlRow.$insert() Returns lReturnFlag' and press Ctrl-I/Cmnd-I to *insert an empty line above*.

Note you can press **Ctrl/Cmnd-+** to increase the font size in the Code Editor.

- With the cursor on the empty line, press "c" to locate the *Calculate* command: the Code Assistant will drop down from the code line and display a list of commands starting with the letter "C".

- In this case, the Calculate command is the first command in the list so, *assuming Calculate is selected,* press the **Return** key to enter the command.
- As soon as you select the Calculate command, the cursor is placed in the code line where a field or variable name is required. If you know the variable name, and in this case the column name, you can enter the text: **iSqlRow.Pic_ID** (this is the row variable and column name containing some data).
- Alternatively, you can use the **Code Assistant:** to do this, type the first letter of the variable, in this case "i", and select the **iSqlRow** variable name from the list that drops down

You can select the variable name with the pointer, or to keep your hands on the keyboard, use the **Up** or **Down Arrow** keys to move up and down in the list and press the **Return** key to confirm your choice, in this case select **iSqlRow.** This variable is contained in the form and was added automatically by the SQL form wizard – when the form is opened the variable will contain the current row of data.

- Having selected the variable name from the Code Assistant (or entered the name manually), type a dot (period), then enter the database column name, in this case, **Pic_ID** – the complete text should be: **iSqlRow.Pic_ID**

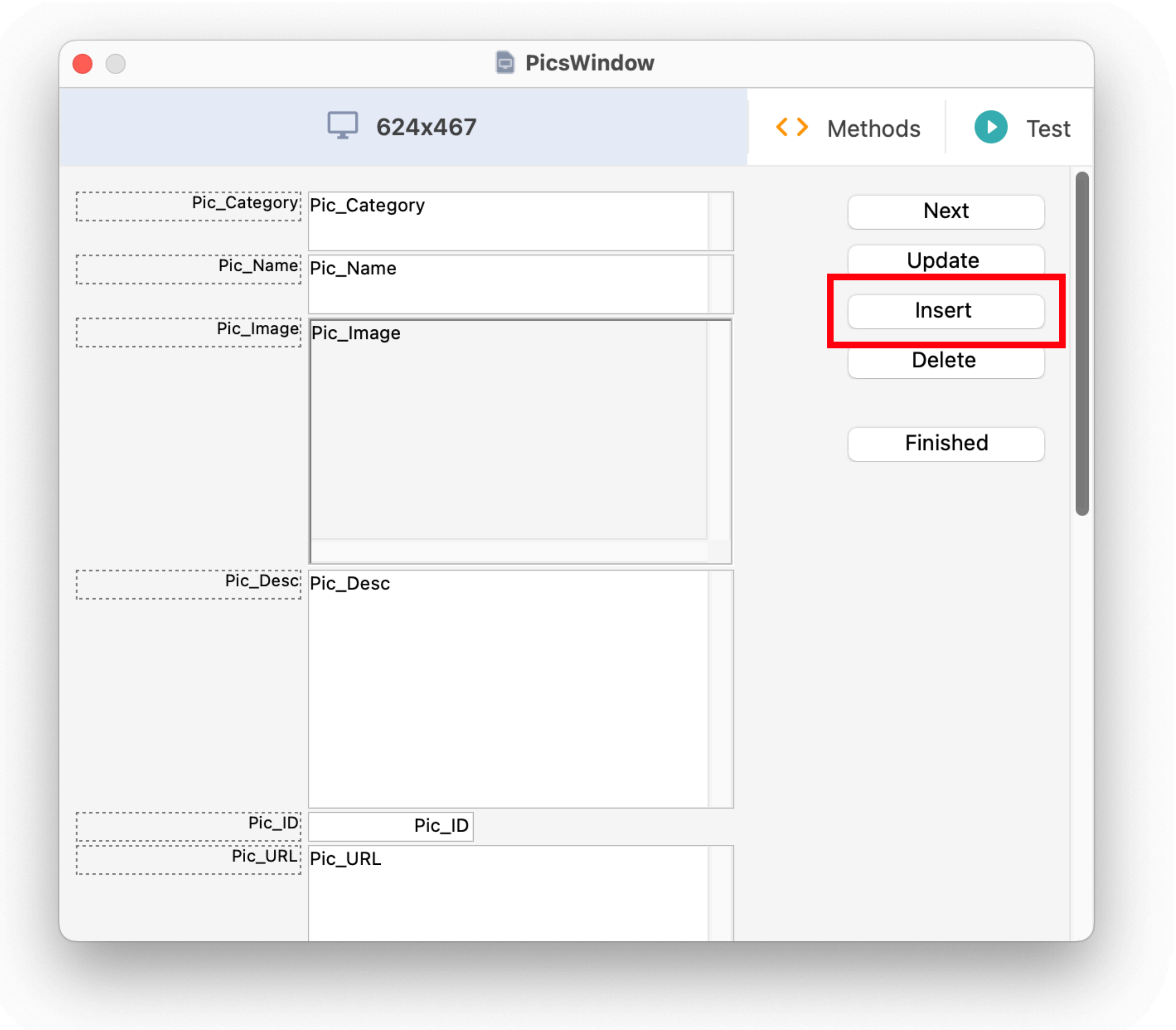
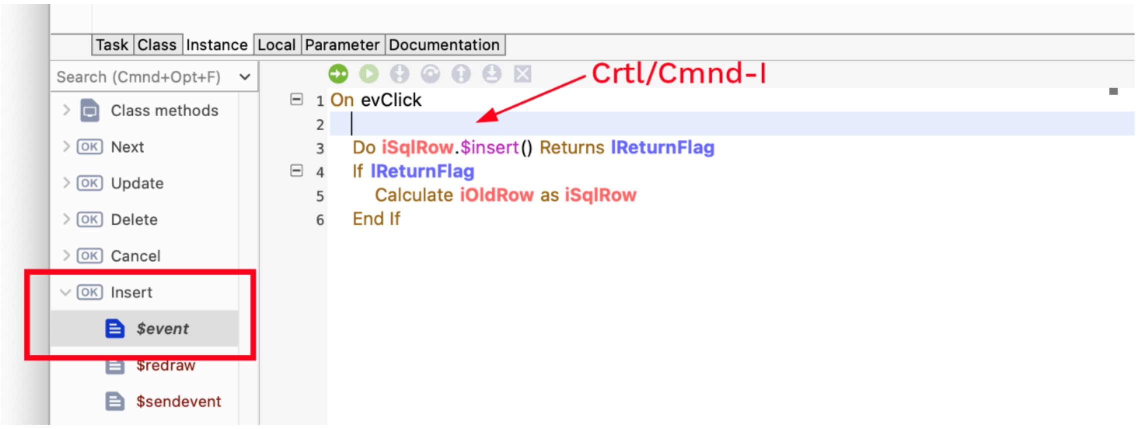
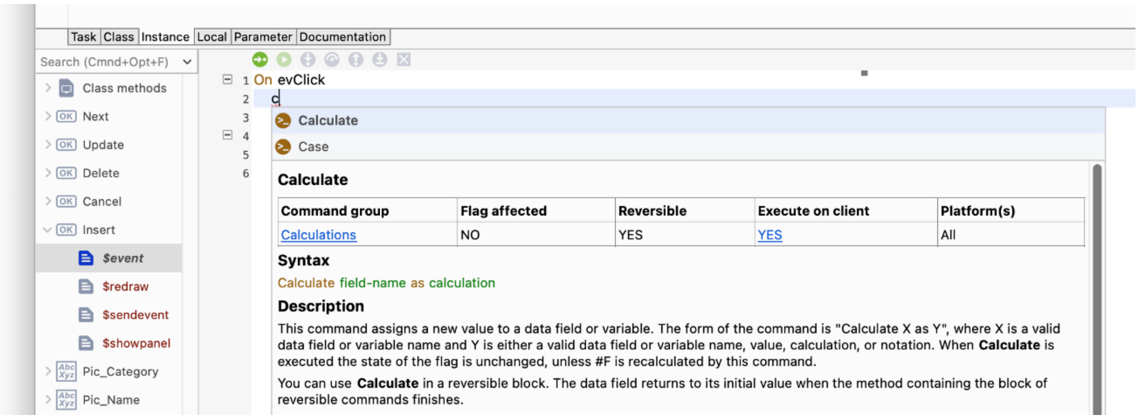


Figure 24:



```
On evClick

  Do iSqlRow.$insert() Returns lReturnFlag
  If lReturnFlag
    Calculate iOldRow as iSqlRow
  End If
```

Figure 25:



```
On evClick
    c
```

Calculate

Command group	Flag affected	Reversible	Execute on client	Platform(s)
Calculations	NO	YES	YES	All

Syntax

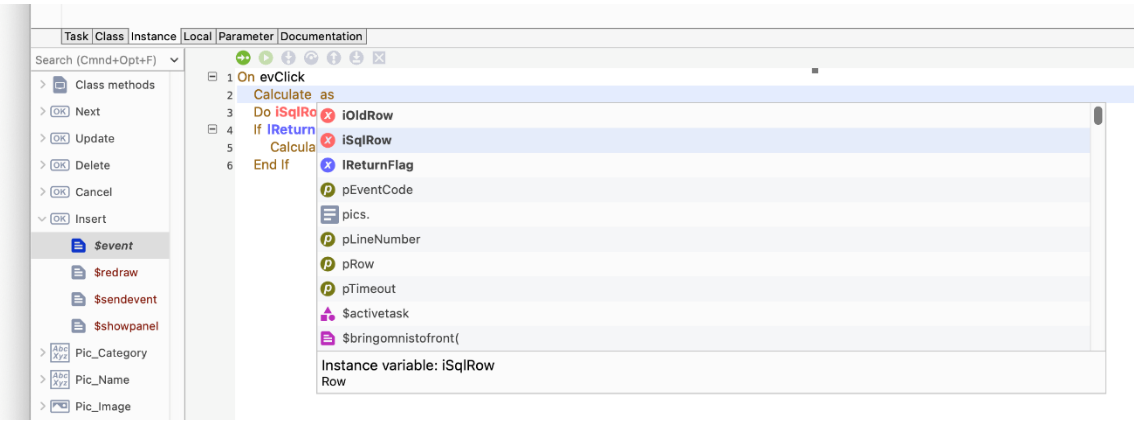
Calculate field-name as calculation

Description

This command assigns a new value to a data field or variable. The form of the command is "Calculate X as Y", where X is a valid data field or variable name and Y is either a valid data field or variable name, value, calculation, or notation. When **Calculate** is executed the state of the flag is unchanged, unless #F is recalculated by this command.

You can use **Calculate** in a reversible block. The data field returns to its initial value when the method containing the block of reversible commands finishes.

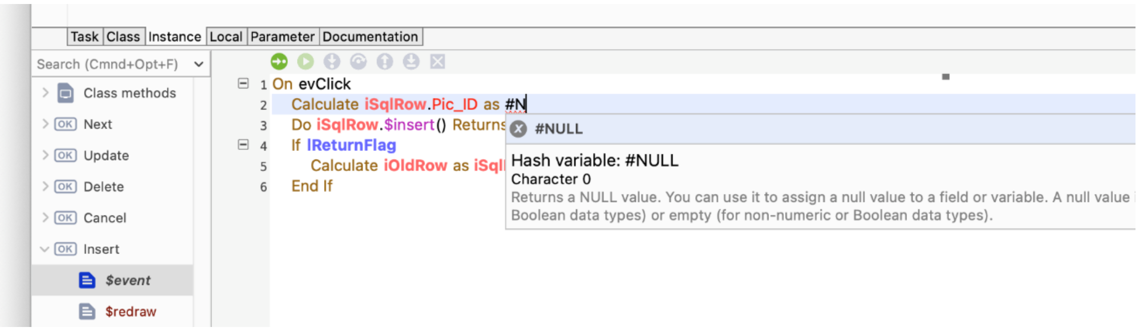
Figure 26:



```
On evClick
    Calculate  as
    Do iSqlRo
    If lReturn
        Calcula
    End If
```

Instance variable: iSqlRow
Row

Figure 27:



```
On evClick
    Calculate iSqlRow.Pic_ID as #N
    Do iSqlRow.$insert() Returns
    If lReturnFlag
        Calculate iOldRow as iSql
    End If
```

Hash variable: #NULL
Character 0
Returns a NULL value. You can use it to assign a null value to a field or variable. A null value i
Boolean data types) or empty (for non-numeric or Boolean data types).

Figure 28:

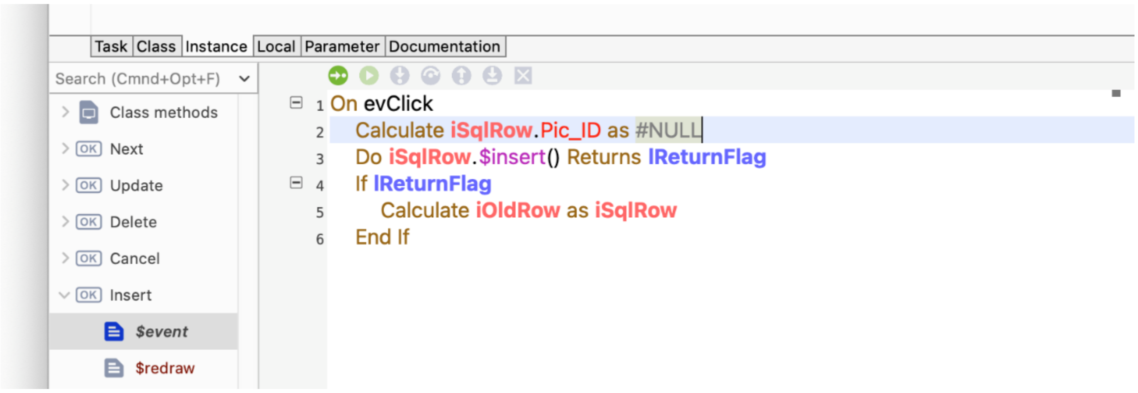
- Press the **Tab** key to move the cursor to the end of the line to enter the *Calculation***,** then enter **#NULL**

To enter #NULL you can type "#" (Shift+3 or Option/Alt+3 or Fn+Option+3 on a Mac keyboard) and the Code Assistant will display all the "hash variables", or in this case, type "#N" and use the **Down Arrow** and **Return** keys to select "#NULL" from the Code Assistant.

#NULL is a so-called "hash variable", a built-in Omnis variable (constant), that represents a NULL value. The line of code you added to the Insert button ensures that whatever value is entered into the field in the open window, just before the row of data is inserted, the value of Pic_ID in the window is set to NULL and SQLite will insert a new value automatically. The complete code line is:

```
Calculate iSqlRow.Pic_ID as #NULL
```

Your method should now look like the following:



```
On evClick
  Calculate iSqlRow.Pic_ID as #NULL
  Do iSqlRow.$insert() Returns lReturnFlag
  If lReturnFlag
    Calculate iOldRow as iSqlRow
  End If
```

Figure 29:

- When you have finished editing the method, close the Method Editor by clicking on the close box, and then save your library using Ctrl/Cmnd-S, click on the **Save** button on the main Omnis toolbar, or select **Save** from the File menu (or if you have enabled **Auto Save** in the File menu, the class will be saved automatically).

Testing a Desktop Form

Now you can test the form you have created to ensure that in runtime mode it performs as expected. To do this:

- Click on the **Test** button in the Design bar to test or open the window class
- Alternatively, you can press **Ctrl/Cmnd-T** to test or open a form or window, or you can Right-click on the window background and select **Open Window**

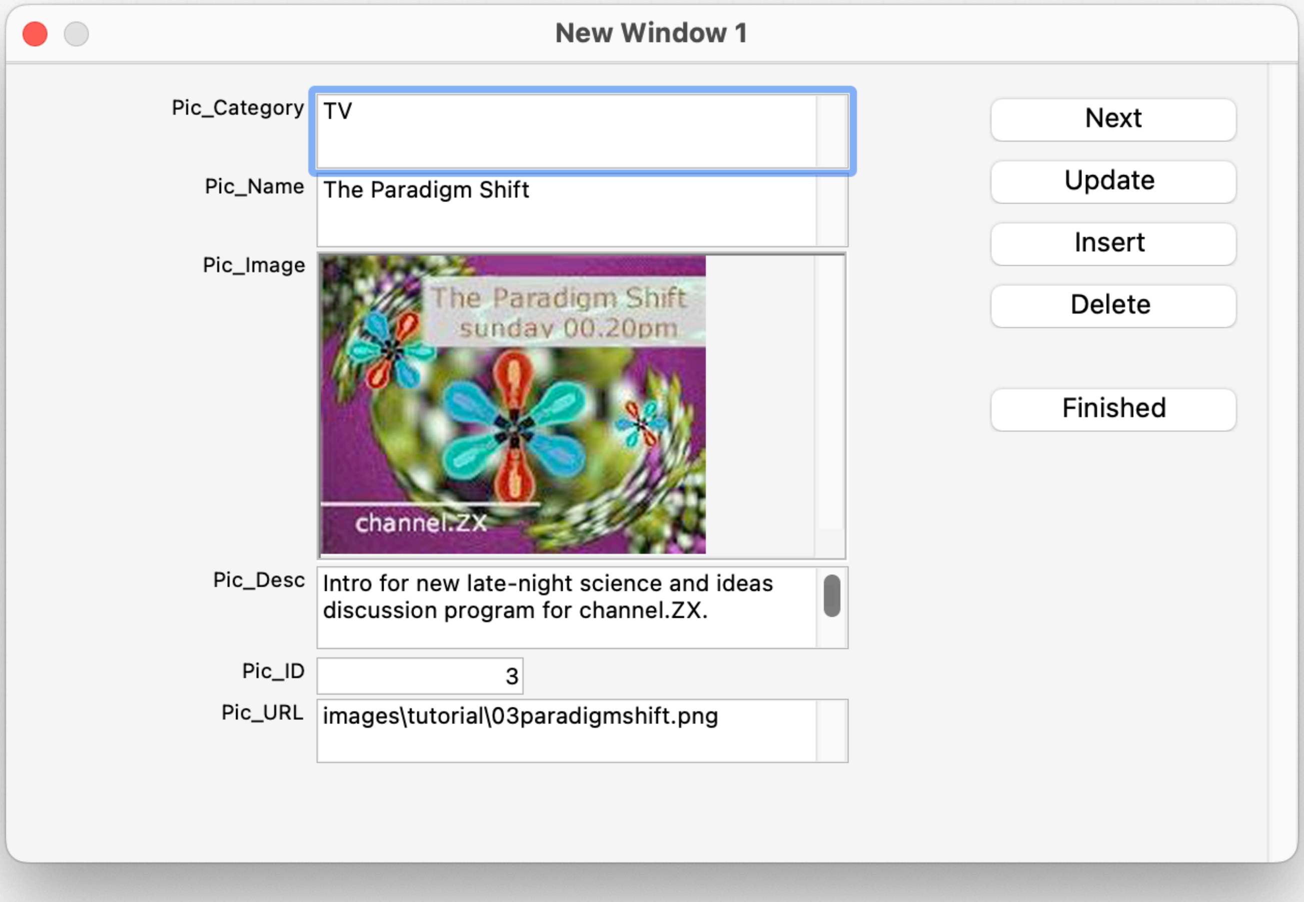
This will create a *window instance* on the screen. The window is opened on top of the design window. Note that when you press Ctrl/Cmnd-T to test your form you must first make sure that the design window for **PicsWindow** is the top window and not some other Omnis window such as the Component Store or Property Manager.

- Press the **Next** button on the open window to see the first row of data in the MyPictures table: click Next again to cycle through the data.

If you don't see any data in your form, check that the database session is open (for example, if you have closed and reopened Omnis, then resumed the tutorial, you will need to open the database session again via the SQL Browser, as described in an earlier section in this tutorial).

- Press the **Next** button to cycle through the other rows of data in the MyPictures table.

When you are developing a form, you can press Ctrl/Cmnd-T at any time to switch between the *design mode* and *runtime* (open mode), and back to design mode again.



Pic_Category TV

Pic_Name The Paradigm Shift

Pic_Image

Pic_Desc Intro for new late-night science and ideas discussion program for channel.ZX.

Pic_ID 3

Pic_URL images\tutorial\03paradigmshift.png

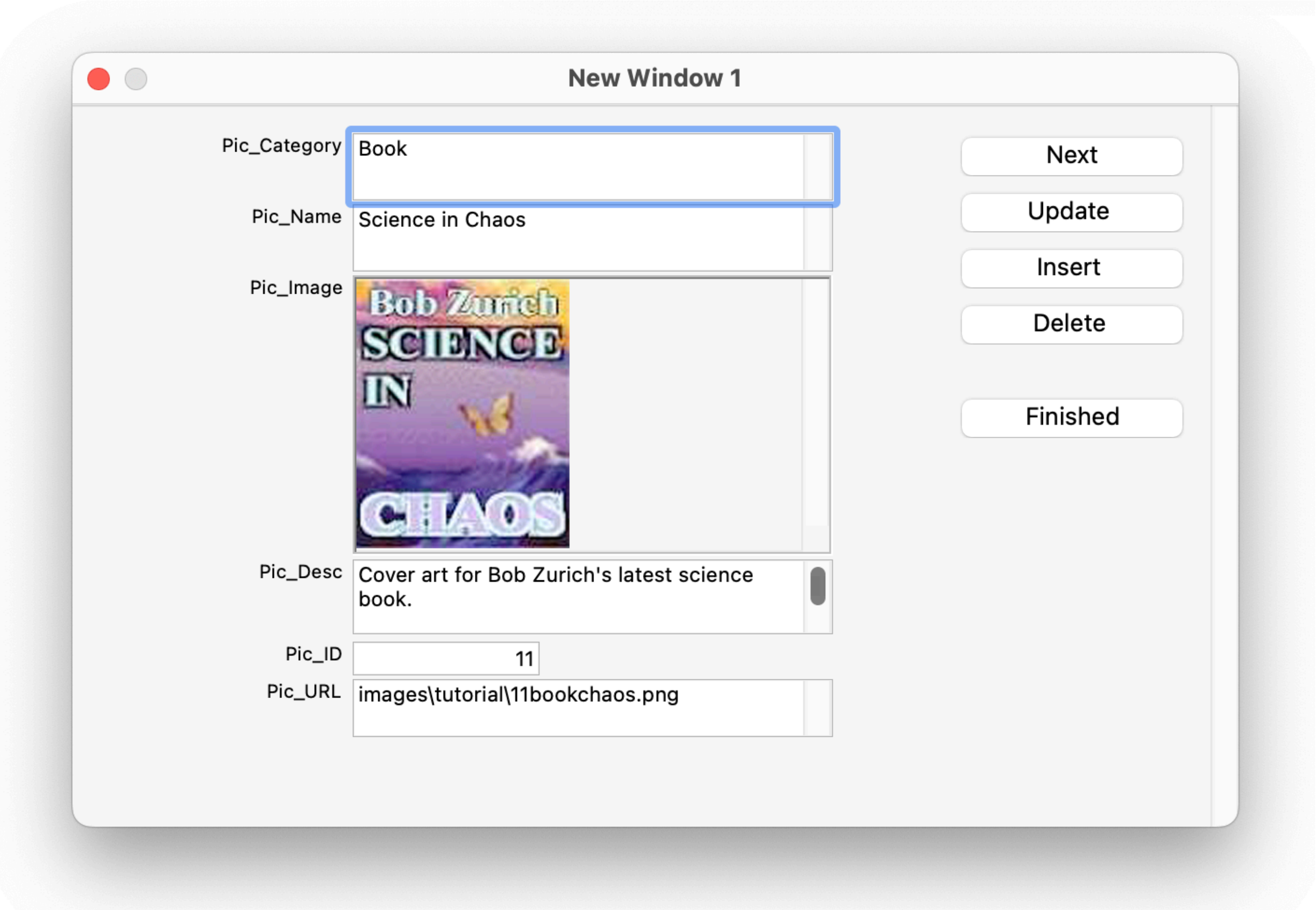
Figure 30:

Inserting some data

Next you can insert a new record in the database. To insert data (a record), ***first enter some data into the fields*** as follows (the precise details are not important for this test, but do insert the PNG image into Pic_Image and insert the correct text in Pic_URL):

Field	**Data to enter...**
Pic_Category	Book
Pic_Name	Science in Chaos
Pic_Image	Insert '11bookchaos.png' from the tutorial folder you downloaded: to do this, use the **Paste from File** option, available from the Edit menu (you might need to press Alt or Option under Windows to show the menu bar including the Edit menu): *you may need to select the PNG image type to see the file.*
Pic_Desc	Cover art for Bob Zurich's latest science book
Pic_ID	**leave this field blank;** a new unique value of Pic_ID will be inserted for you automatically
Pic_URL	**images\tutorial\11bookchaos.png** (this is not required for the desktop window, but this information is required when viewing the data in a web browser created in the next part of the tutorial)

- When you have entered the new data record, click on the ***Insert*** button to insert or "save" the record.



Pic_Category: Book

Pic_Name: Science in Chaos

Pic_Image

Pic_Desc: Cover art for Bob Zurich's latest science book.

Pic_ID: 11

Pic_URL: images\tutorial\11bookchaos.png

Figure 31:

Important note: Your form must be in open/runtime mode to test it and edit/enter data.

- After you have clicked (once only) on the Insert button, click on Next until you cycle through to the new record you just entered.
- Close the pics window and the design window.

Creating a Web Form from scratch

*In this section, you will build a remote form "from scratch" starting with a blank form and adding controls one by one. Alternatively, you could use the **SQL JavaScript Form Wizard** which automates the whole process and is much quicker. So, to save yourself time, you may like to go straight to the 'Creating a Web Form using a wizard' section, ignoring the following section.*

In this section of the Tutorial, you are going to create a JavaScript-based web form (called a "Remote form class" in Omnis) to browse your database in a desktop web browser. You can deploy the remote form on the Omnis App Server* & Web Server to allow anyone to look at your picture database in a browser on a desktop computer or mobile device (*this would require a Web license to run your app, which is available for free with the Community Edition, or for a license fee with other editions).

Remember, if you have quit and restarted Omnis Studio, then you will need to reopen your database session, described in the 'Opening a Database Session' section, to complete this section.

To create a remote form from scratch (i.e. not using a wizard):

- Click the **PICS** library in the Folders tree list on the Studio Browser window to show the New class options.
- Click on the **New Class** option in the Studio Browser, then the **RemoteForm** option; name the new class **PicsWebform** and press Return.

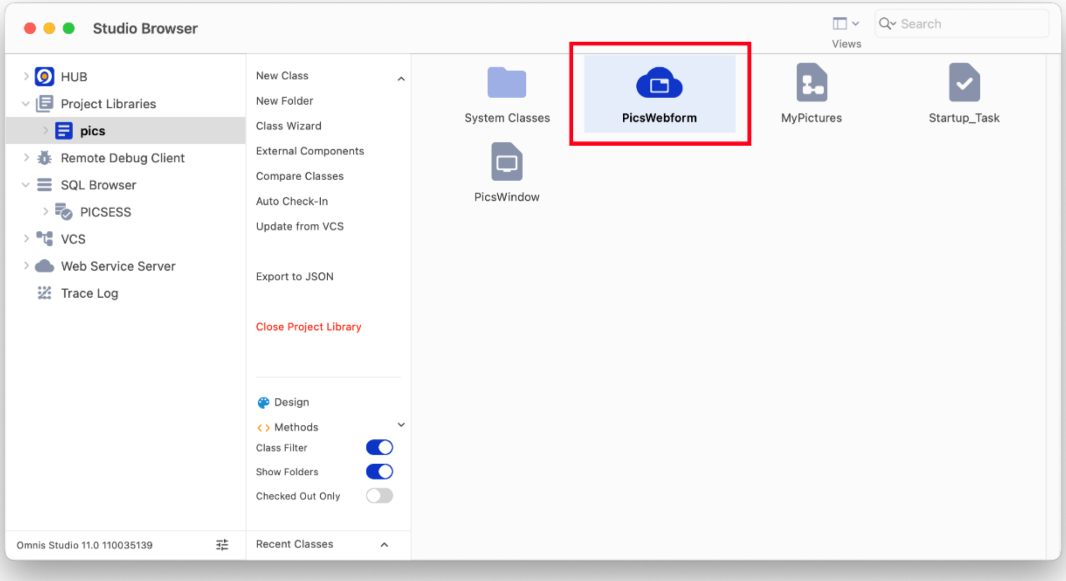


Figure 32:

The new Remote Form will appear in the Studio Browser (note that if you are using the Community Edition there will also be a Remote_Task class).

1. Double-click on **PicsWebform** to open it in design mode.

When you design a remote form the **Component Store** will open automatically docked to the left side of the Remote form editor. It contains 40 or so ready-made JavaScript components, arranged in functional groups, which you can drag and drop onto your remote form.

The remote form has two layout breakpoints, set to **320** and **768,** which are shown in the Design bar of the Remote form editor; there are also options to show the **Methods** for the form and to **Test** the form in a web browser (you can use these later). The layout breakpoints allow the layout of the form to change at runtime as the size of the client browser changes (mobile or desktop). *If necessary, enlarge the form design window and click* on **768** to select it.

You will add some fields and other controls to the design layout for the **768 layout breakpoint,** and later in this section you can adjust the layout for the 320 breakpoint, so your form can be displayed on phones as well.

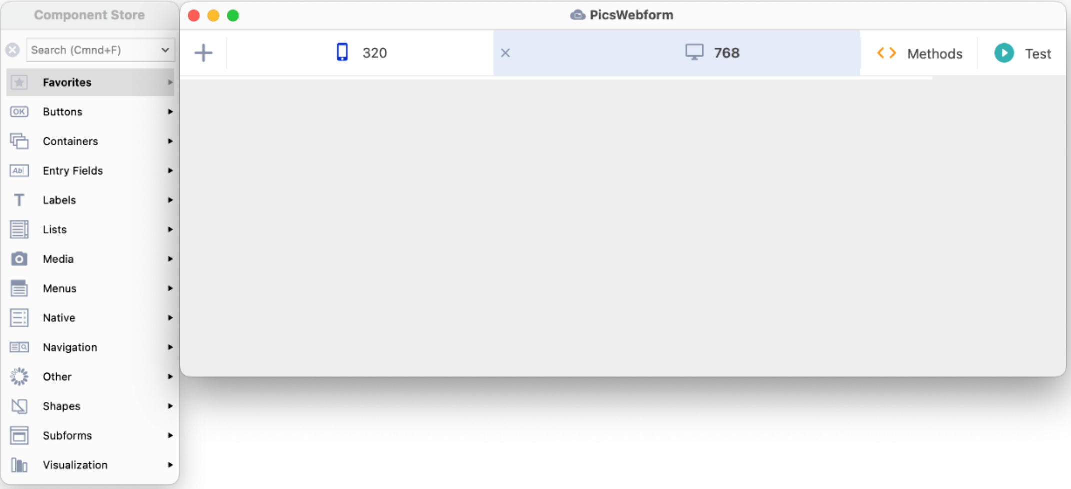


Figure 33:

Adding Fields to a Web Form

You can add fields to your form by dragging icons from the Component Store and dropping them onto the form.

- Press **F3/Cmnd-3** to bring the Component Store to the top, click on the **Entry Fields** group and locate the **Entry Field** control.

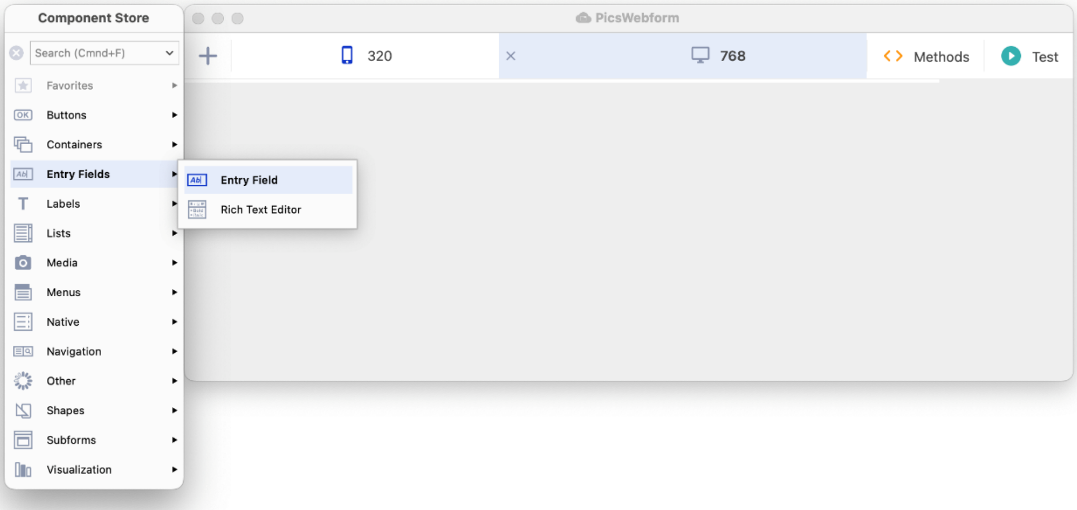


Figure 34:

- Drag the **Entry Field** control from the Component Store *and drop it onto your form,* somewhere near the top-left corner (as shown below).

An **Entry Field** control is placed on your form and assigned a default **name,** but you can change it using the **Property Manager.**

- Press **F6/Cmnd-6** to bring the Property Manager to the top.

Hiding and Showing Properties

The **Advanced** option in the Property Manager will be **disabled** (off) when you first launch Omnis, so the Property Manager will display a simpler subset of properties: for the purposes of the tutorial, all the properties you need should be visible with Advanced disabled.

Once you are familiar with Omnis, as well as the components and their properties, you might like to enable the **Advanced** option to show all properties whenever the Property Manager is opened.

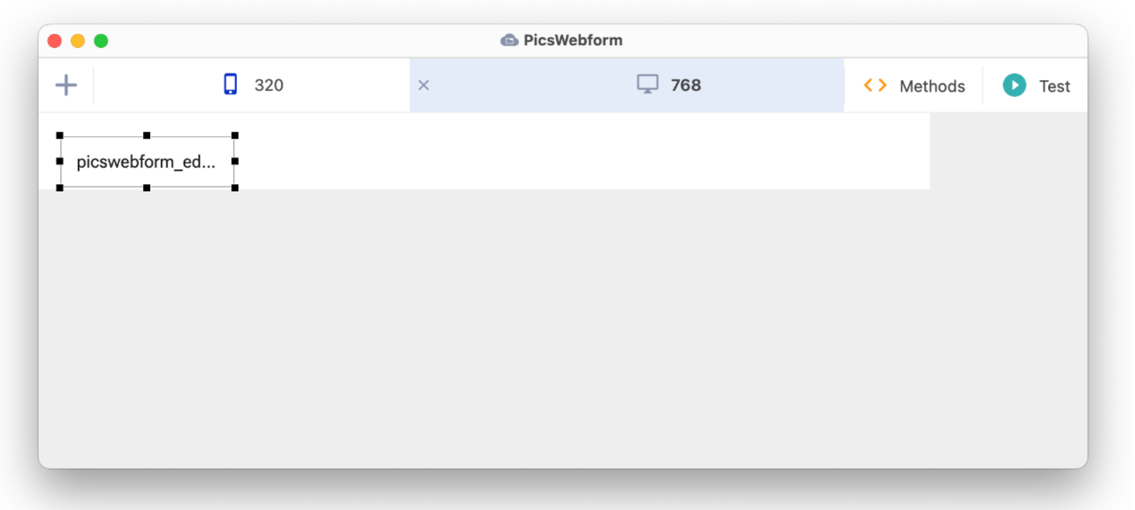
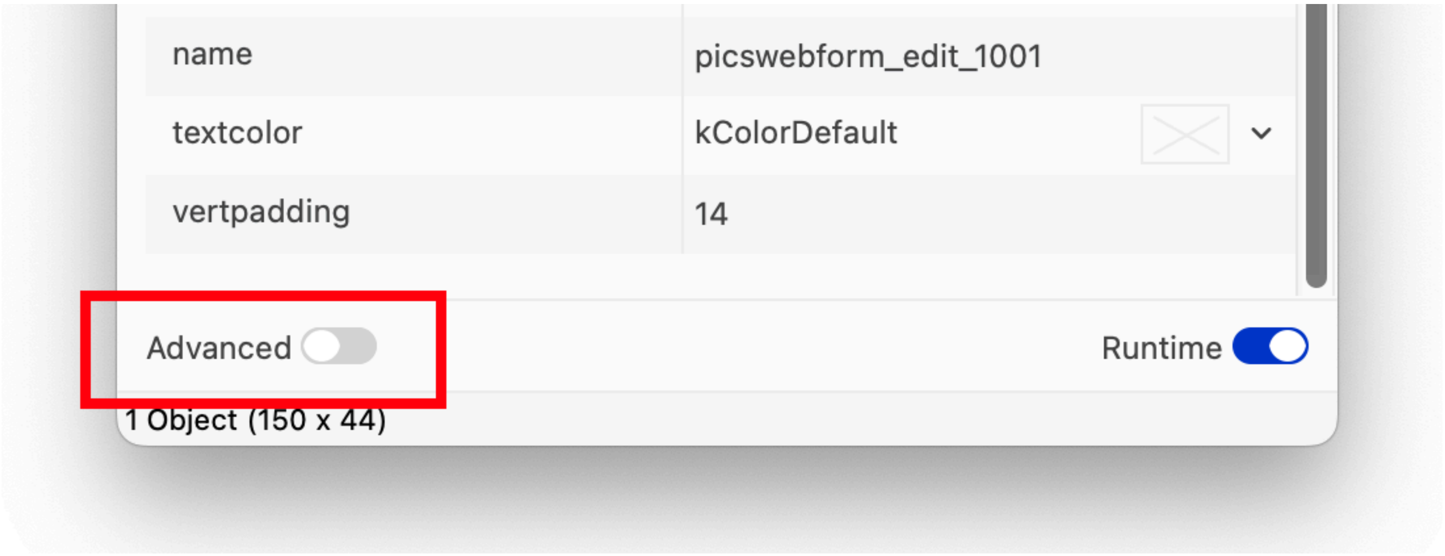


Figure 35:



name	picswebform_edit_1001
textcolor	kColorDefault
vertpadding	14

Advanced ○

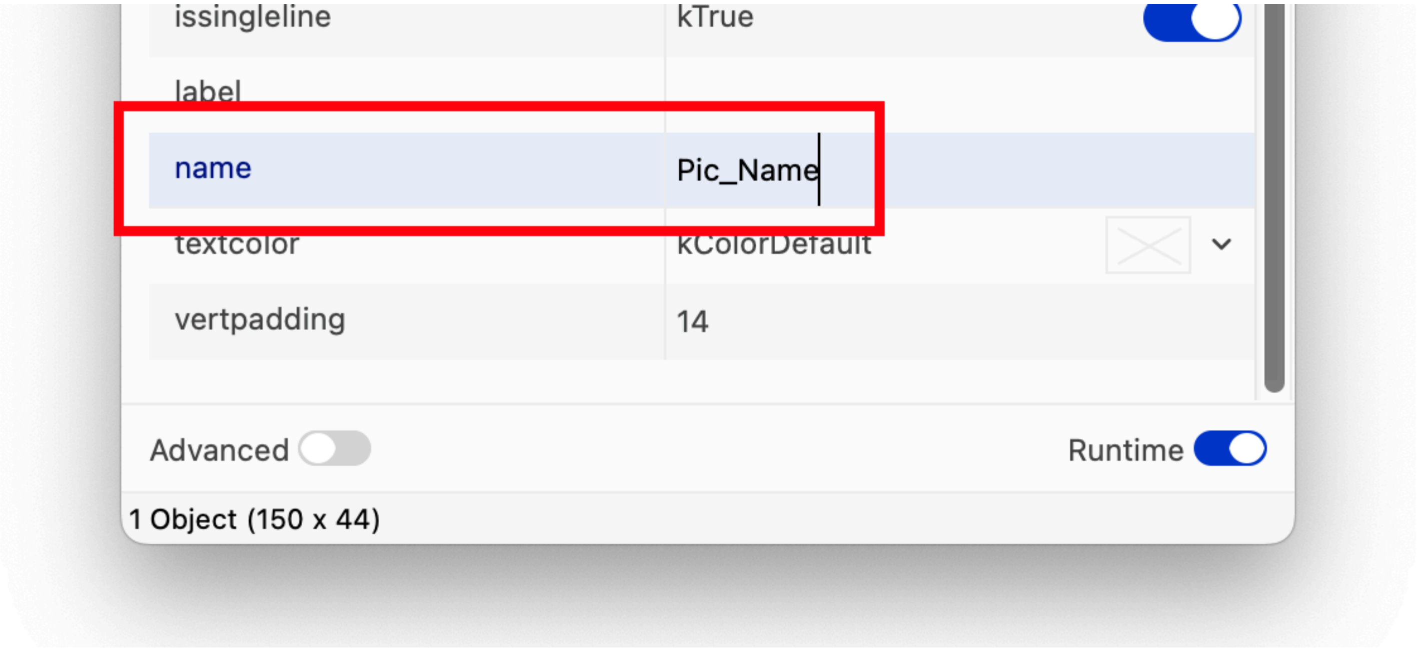
Runtime ●

1 Object (150 x 44)

Figure 36:

Changing a Property

- Locate the **name** property in the list of properties in the Property Manager (you may need to scroll the list).
- Click in the cell next to the **name** property, delete the name assigned by Omnis automatically and enter **Pic_Name**



issingleline	kTrue
label	
name	Pic_Name
textcolor	kColorDefault
vertpadding	14

Figure 37:

- In the remote form, drag the right end of the entry field to make it wider (if necessary, drag the border of the remote form itself to resize it).

Your form should look something like the following.

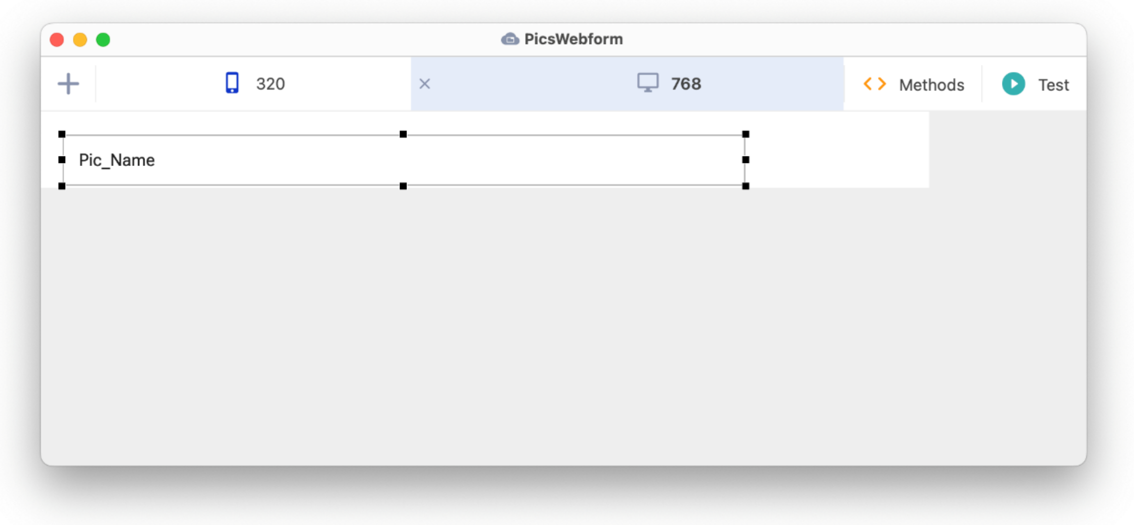


Figure 38:

The minimum height of the current layout is set to 2 pixels below the bottom of the lowest control on the form and this is adjusted automatically as you add or move components. The minimum height is stored in the $layoutminheight remote form property for each layout breakpoint, while the padding is stored in $layoutpadding (default is 2). The minimum height of the remote form is shown as the white area enclosing all the controls on the form.

Adding Code and Further Fields to a Form

Next you need to add a **dataname** or *variable* name to the edit field you have created to associate the entry field with a column in your database.

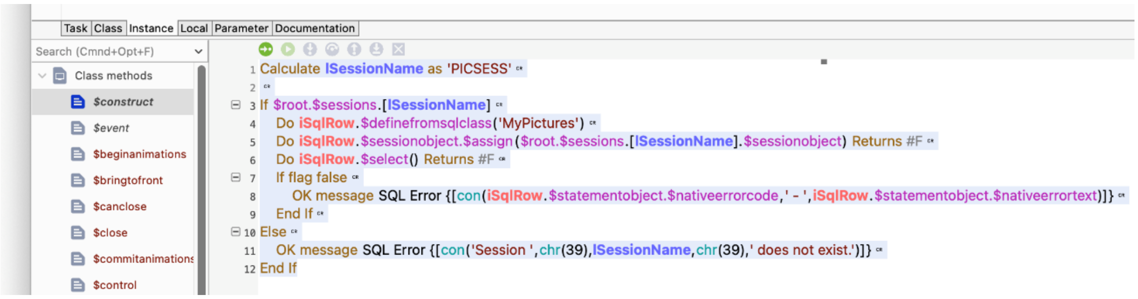
Before you can add a dataname to the edit control, you need to add a **Row variable** to the form that will link your form to the MyPictures table in your database. You can add just the variable alone, or you can add or copy the code needed in the form, which contains the required variables, and then add the variable(s) directly in the code.

If you are using the Community Edition of Omnis Studio, you can jump to the 'Pasting in the code...' section below and enter the code and variables manually.

Otherwise, if you are using the Professional Edition of Omnis Studio, or the free trial, you can copy the code from the PicsWindow (window class) you created in the previous section of this tutorial, or if you did not complete that section you can grab the code from one of the pics.lbs or pics2.bs library in the 'final' folder.

To copy code from a window class: (not for Community edition users)

- Press F2/Cmnd-2 to bring the Studio Browser to the top and double-click the **PicsWindow** class to open it in design mode.
- Double-click somewhere *on the background of PicsWindow* (*not on a field*) to open the Omnis **Method Editor** for the form.
- Make sure the $construct method is visible and selected (see below), then select all the lines of code in the $construct method and Copy them (press **Ctrl/Cmnd-C** or use the Edit>>Copy menu option).



```
Calculate lSessionName as 'PICSESS'

If $root.$sessions.[lSessionName]
  Do iSqlRow.$definefromsqlclass('MyPictures')
  Do iSqlRow.$sessionobject.$assign($root.$sessions.[lSessionName].$sessionobject) Returns #F
  Do iSqlRow.$select() Returns #F
  If flag false
    OK message SQL Error {[con(iSqlRow.$statementobject.$nativeerrorcode,' - ',iSqlRow.$statementobject.$nativeerrortext)]}
  End If
Else
  OK message SQL Error {[con('Session ',chr(39),lSessionName,chr(39),' does not exist.')]}
End If
```

Figure 39:

- Close the Method Editor and close the PicsWindow class.
- Double-click on the **PicsWebform** remote form to open it in design mode and ***double-click on the background*** of the remote form to open the **Method Editor** for this class (not the Pic_Name entry field).
- Make sure the ***$construct method is visible and selected*** under Class methods (on the left), then click on the first line in the right pane of the Code Editor and Paste the code you copied

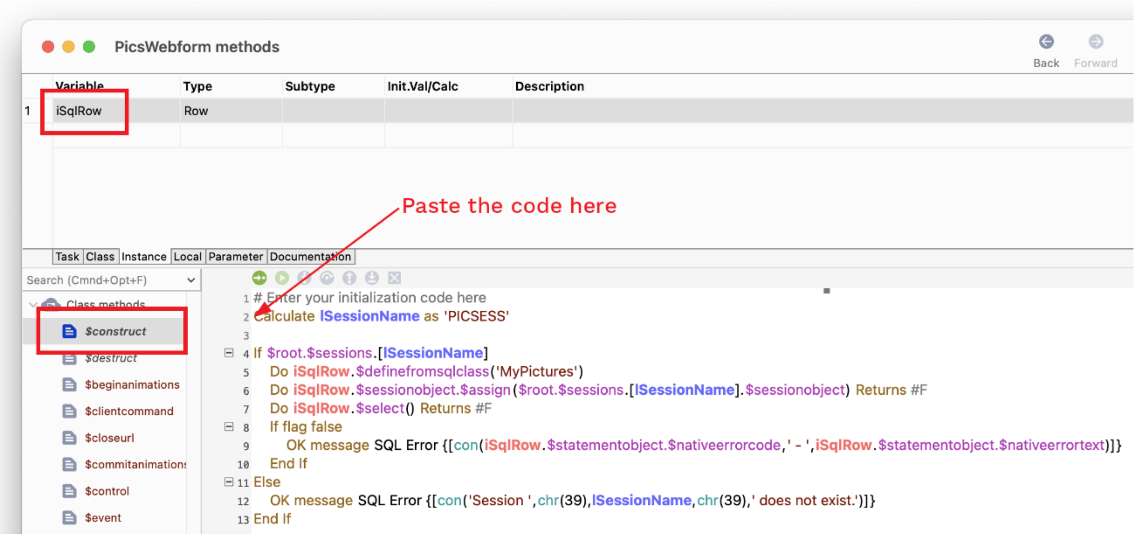
When you paste a method or some code containing any variables *from one class to another* in Omnis, the variables and their definitions are also copied across. Therefore, you will notice that in the Variable pane, Omnis has created the **iSqlRow** instance variable (under the Instance tab) and the **lSessionName** local variable (under the Local tab).

Pasting in the code

If you did not create the PicsWindow window class (or are using the Community edition), you can copy the code text from this page (using the Copy icon, top right of the code box), and create the variables yourself.

- Copy the following code:

```
Calculate lSessionName as 'PICSESS'
If $root.$sessions.[lSessionName]
  Do iSqlRow.$definefromsqlclass('MyPictures')
  Do iSqlRow.$sessionobject.$assign($root.$sessions.[lSessionName].$sessionobject) Returns #F
  Do iSqlRow.$select() Returns #F
```



	Variable	Type	Subtype	Init.Val/Calc	Description
1	iSqlRow	Row			

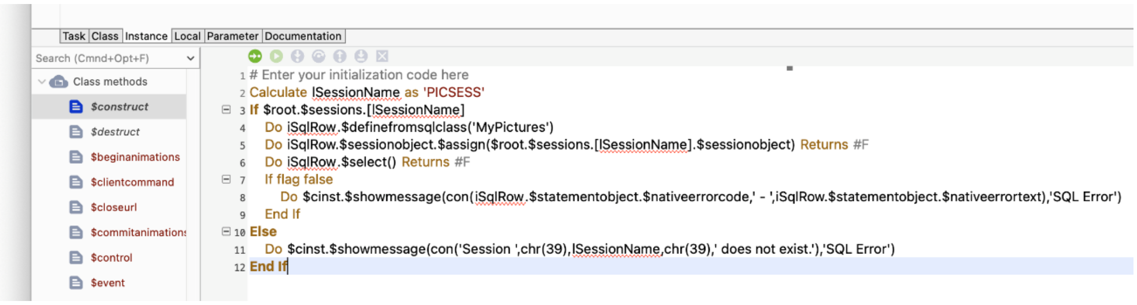
```
# Enter your initialization code here
Calculate lSessionName as 'PICSESS'

If $root.$sessions.[lSessionName]
  Do iSqlRow.$definefromsqlclass('MyPictures')
  Do iSqlRow.$sessionobject.$assign($root.$sessions.[lSessionName].$sessionobject) Returns #F
  Do iSqlRow.$select() Returns #F
  If flag false
    OK message SQL Error {[con(iSqlRow.$statementobject.$nativeerrorcode,' - ',iSqlRow.$statementobject.$nativeerrortext)]}
  End If
Else
  OK message SQL Error {[con('Session ',chr(39),lSessionName,chr(39),' does not exist.')]}
End If
```

Figure 40:

```
  If flag false
    Do $cinst.$showmessage(con(iSqlRow.$statementobject.$nativeerrorcode,' - ',iSqlRow.$statementobject.$nativ
  End If
Else
  Do $cinst.$showmessage(con('Session ',chr(39),lSessionName,chr(39),' does not exist.'),'SQL Error')
End If
```

Next you need to paste the code in the correct place in the PicsWebform.

- Double-click on the **PicsWebform** remote form to open it in design mode (if it's not already open)
- ***Double-click on the background*** of the remote form (not the Pic_Name entry field) to open the **Method Editor** for the class.
- Make sure the ***$construct method is visible and selected*** under Class methods (on the left, see below), then click on the first line in the right pane of the Code Editor and paste the code text you copied



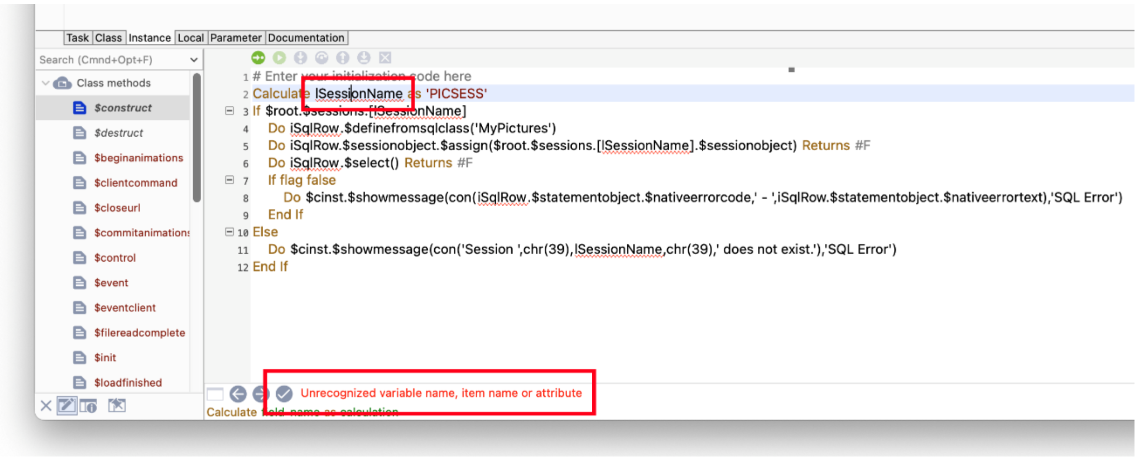
```
# Enter your initialization code here
Calculate lSessionName as 'PICSESS'
If $root.$sessions.[lSessionName]
  Do iSqlRow.$definefromsqlclass('MyPictures')
  Do iSqlRow.$sessionobject.$assign($root.$sessions.[lSessionName].$sessionobject) Returns #F
  Do iSqlRow.$select() Returns #F
  If flag false
    Do $cinst.$showmessage(con(iSqlRow.$statementobject.$nativeerrorcode,' - ',iSqlRow.$statementobject.$nativeerrortext),'SQL Error')
  End If
Else
  Do $cinst.$showmessage(con('Session ',chr(39),lSessionName,chr(39),' does not exist.'),'SQL Error')
End If
```

Figure 41:

At this stage the code text contains the variable names (for **lSessionName** and **iSqlRow)** but they need to be defined. You will notice that since the code contains undefined variables it is not color coded, but you can fix this next. To do this:

- Click anywhere within the **lSessionName** name.

- At the bottom of the Code Editor window, click the **Fix** button (the button with Check mark icon), which opens the Create Variable dialog.

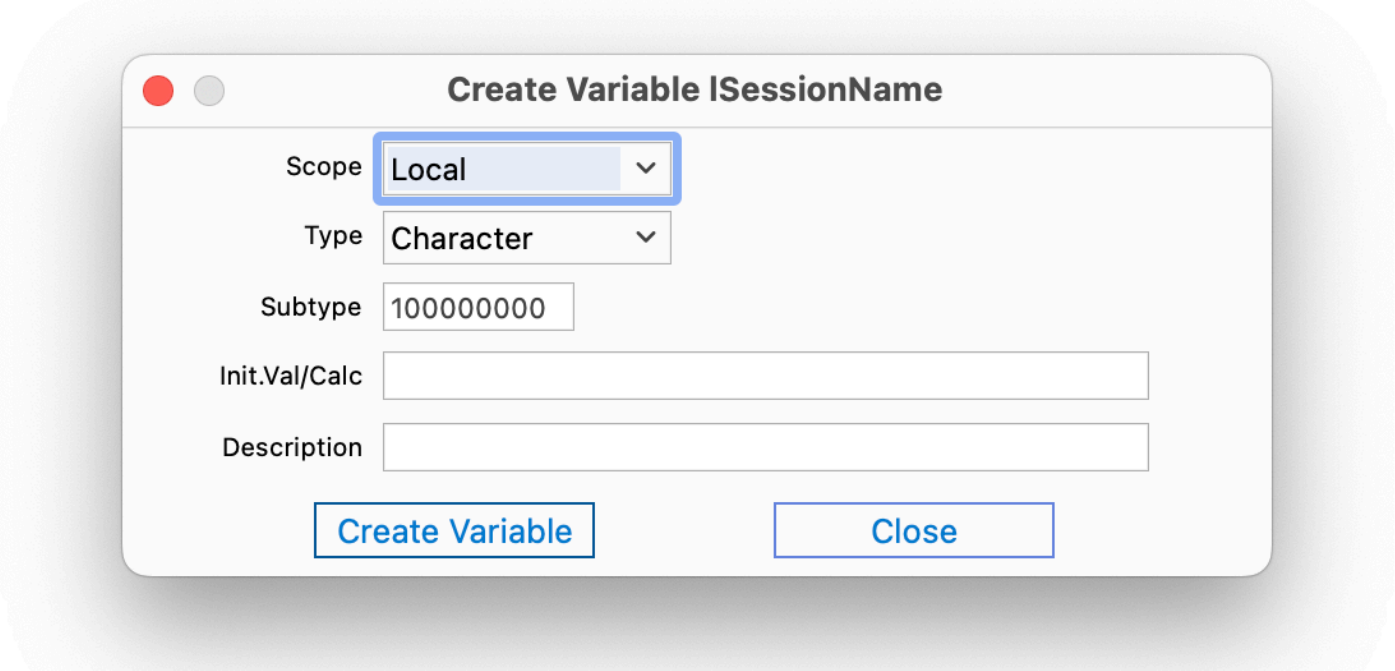


```
# Enter your initialization code here
Calculate lSessionName as 'PICSESS'
If $root.$sessions.[lSessionName]
   Do iSqlRow.$definefromsqlclass('MyPictures')
   Do iSqlRow.$sessionobject.$assign($root.$sessions.[lSessionName].$sessionobject) Returns #F
   Do iSqlRow.$select() Returns #F
   If flag false
      Do $cinst.$showmessage(con(iSqlRow.$statementobject.$nativeerrorcode,' - ',iSqlRow.$statementobject.$nativeerrortext),'SQL Error')
   End If
Else
   Do $cinst.$showmessage(con('Session ',chr(39),lSessionName,chr(39),' does not exist.'),'SQL Error')
End If
```

Unrecognized variable name, item name or attribute

Calculate field name as calculation

Figure 42:



Create Variable lSessionName

Scope: Local

Type: Character

Subtype: 100000000

Init.Val/Calc:

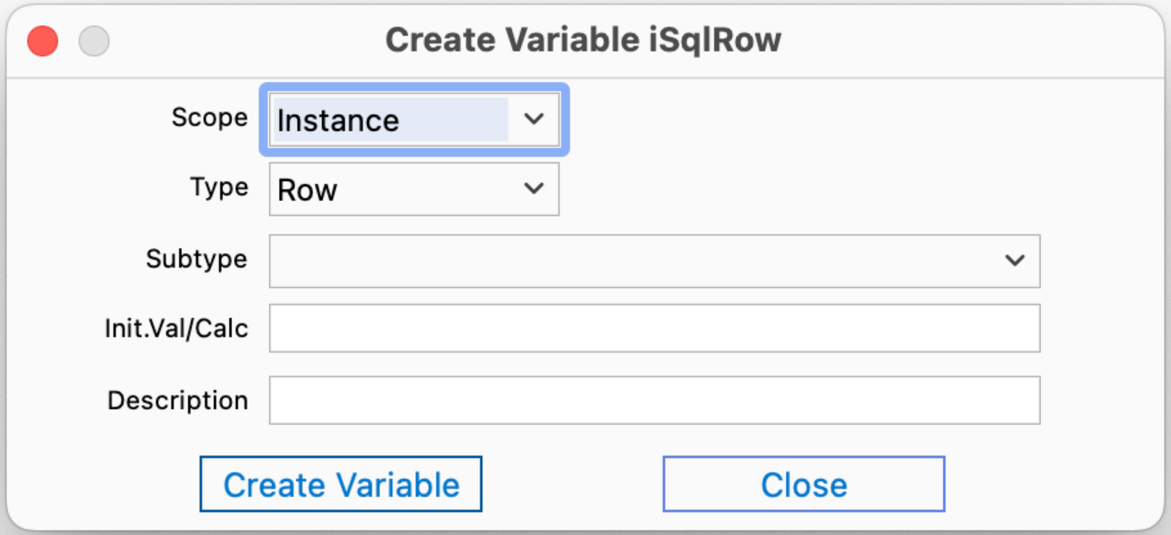
Description:

Figure 43:

In the **Create Variable** dialog for **lSessionName,** ensure that **Local** is selected for Scope, **Character** for Type, and click on **Create Variable** to create the lSessionName local variable. Omnis chooses the scope and type automatically based on the name and context of the code.

The lSessionName is now defined and should have its correct syntax color coding in the Code Editor.

- Next click anywhere within the **iSqlRow** variable name in the code
- At the bottom of the Code Editor window, click on the **Fix** button, and define the **iSqlRow** variable with Instance as the Scope and Row as the Type, and click on Create Variable:



Create Variable iSqlRow

Scope: Instance

Type: Row

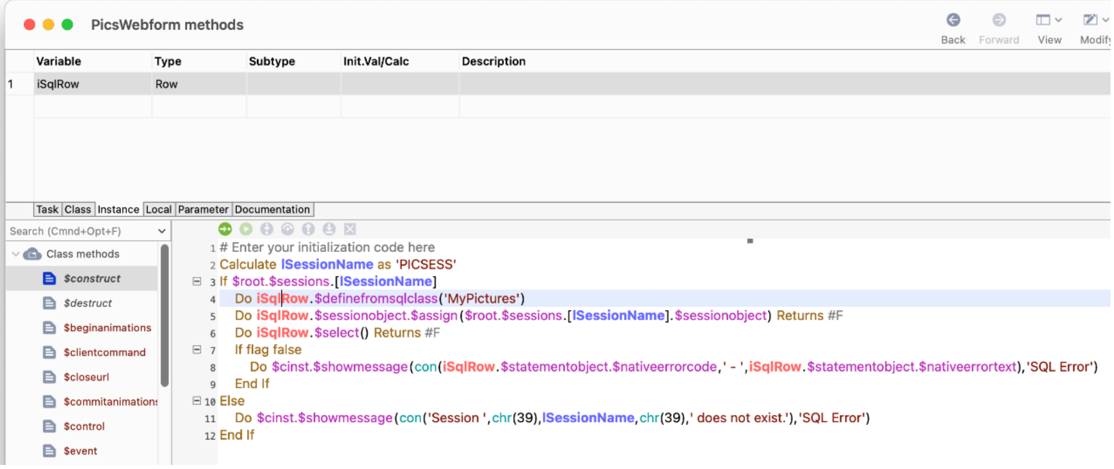
Subtype:

Init.Val/Calc:

Description:

Figure 44:

Now that you have defined the variables, the **iSqlRow** and **lSessionName** variables should appear under their respective tabs (Instance and Local) in the Variables pane, and your code should look like the following, including the syntax color coding:



PicsWebform methods

	Variable	Type	Subtype	Init.Val/Calc	Description
1	iSqlRow	Row			

```
# Enter your initialization code here
Calculate lSessionName as 'PICSESS'
If $root.$sessions.[lSessionName]
  Do iSqlRow.$definefromsqlclass('MyPictures')
  Do iSqlRow.$sessionobject.$assign($root.$sessions.[lSessionName].$sessionobject) Returns #F
  Do iSqlRow.$select() Returns #F
  If flag false
    Do $cinst.$showmessage(con(iSqlRow.$statementobject.$nativeerrorcode,' - ',iSqlRow.$statementobject.$nativeerrortext),'SQL Error')
  End If
Else
  Do $cinst.$showmessage(con('Session ',chr(39),lSessionName,chr(39),' does not exist.'),'SQL Error')
End If
```

Figure 45:

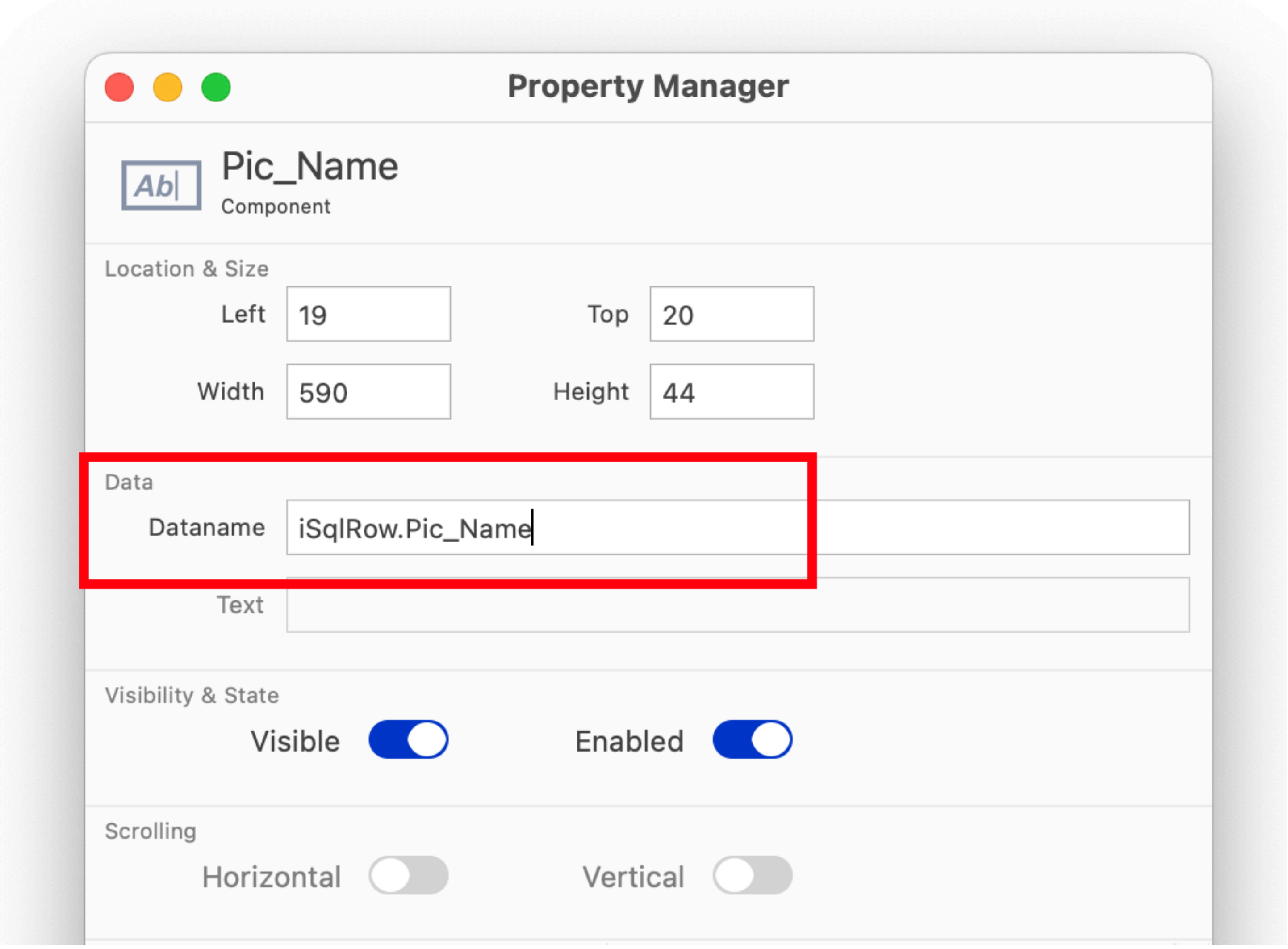
So what does this code do? The $construct method will be executed when the remote form is opened (in the client web browser), so you can use this method to initialize the form and perform any other functions. The three lines of code starting 'Do iSqlRow…' inside the first If statement in the $construct method do the following:

- The first line defines the row variable **iSqlRow** based on the **MyPictures** Schema class in your library (using the $definefromsqlclass method), which itself links to the table in your SQLite database.
- The next line of code creates a session object based on the session template called PICSESS you created previously in this tutorial (using the $sessionobject property).
- The third line of code performs a SELECT on your database which creates a results set of data (using the $select method).

The other code in the $construct method is for handling errors.

- Close the Method Editor; your PicsWebform should still be open
- Click on the **Pic_Name** field in your PicsWebform to select it, then press **F6/Cmnd-6** to open the **Property Manager** (or bring it to the top).
- Locate the **Dataname** property in the Property Manager which should appear in the top panel of the Property Manager under the **Data** heading
- Enter the dataname: **iSqlRow.Pic_Name**

You can type the variable name and column name in full, or to use the **Code Assistant,** type the first letter of the variable name, in this case "i", and select the variable name from the popup, then add a "dot" (full stop/period) and type the column name Pic_Name.



Left: 19
Top: 20
Width: 590
Height: 44

Data
Dataname: iSqlRow.Pic_Name
Text:

Figure 46:

Pic_Name is a column in the iSqlRow variable that is linked to the Pic_Name column in the MyPictures table in the database.

You need to create some other fields in the form, but rather than creating them from scratch you can copy the Pic_Name field and change the properties of each field.

- Click on the **Pic_Name** field and copy it using **Ctrl/Cmnd-C** then **Ctrl/Cmnd-V** to paste a copy (or use the Edit>>Copy / Paste menu options). Alternatively, you can hold down the Ctrl key (on Windows) or Alt/Option key (on macOS), *click inside* the Pic_Name field and *drag it to a new position* to duplicate it
- *With the NEW field selected,* press F6/Cmnd-6 to open the Property Manager (or bring it to the top), and change the following properties of the new field:

  dataname = iSqlRow.Pic_Category
  name = Pic_Category

- Make the **Pic_Category** field narrower and move it to the right, so its top edge is level with the Pic_Name field.

The **Position Assistance** (shown as dotted lines, as above) will help you line up the controls by their top edges; when the dotted line appears you can release the control and it will snap into position.

- Select the Pic_Category field, copy and paste it onto the form (using Ctrl/Cmnd-C then Ctrl/Cmnd-V), and change the properties of the new field using the Property Manager (F6/Cmnd-6) as follows:

  dataname = iSqlRow.Pic_Desc
  name = Pic_Desc
  issingleline = disable (or turn off) the **issingleline** property so the field will allow multiple lines of text, which is required for longer text content; this sets the property $issingleline to kFalse

- Make the **Pic_Desc** field a little deeper and place it under the Pic_Category field; the Position Assistance (dotted lines & auto snap) will help you line up the left edges of the controls.

Now you need to create a further field.

- Copy and paste the **Pic_Category** field again (using Ctrl/Cmnd-C then Ctrl/Cmnd-V), move it down and place it under the Pic_Desc field

You can use the Position Assistance to line up the left edges of the fields and to distribute them equally in a vertical direction.

- Change the properties of the new field, again using the Property Manager (F6/Cmnd-6), as follows:

  dataname = iSqlRow.Pic_ID
  name = Pic_ID

Your form should look something like the following:

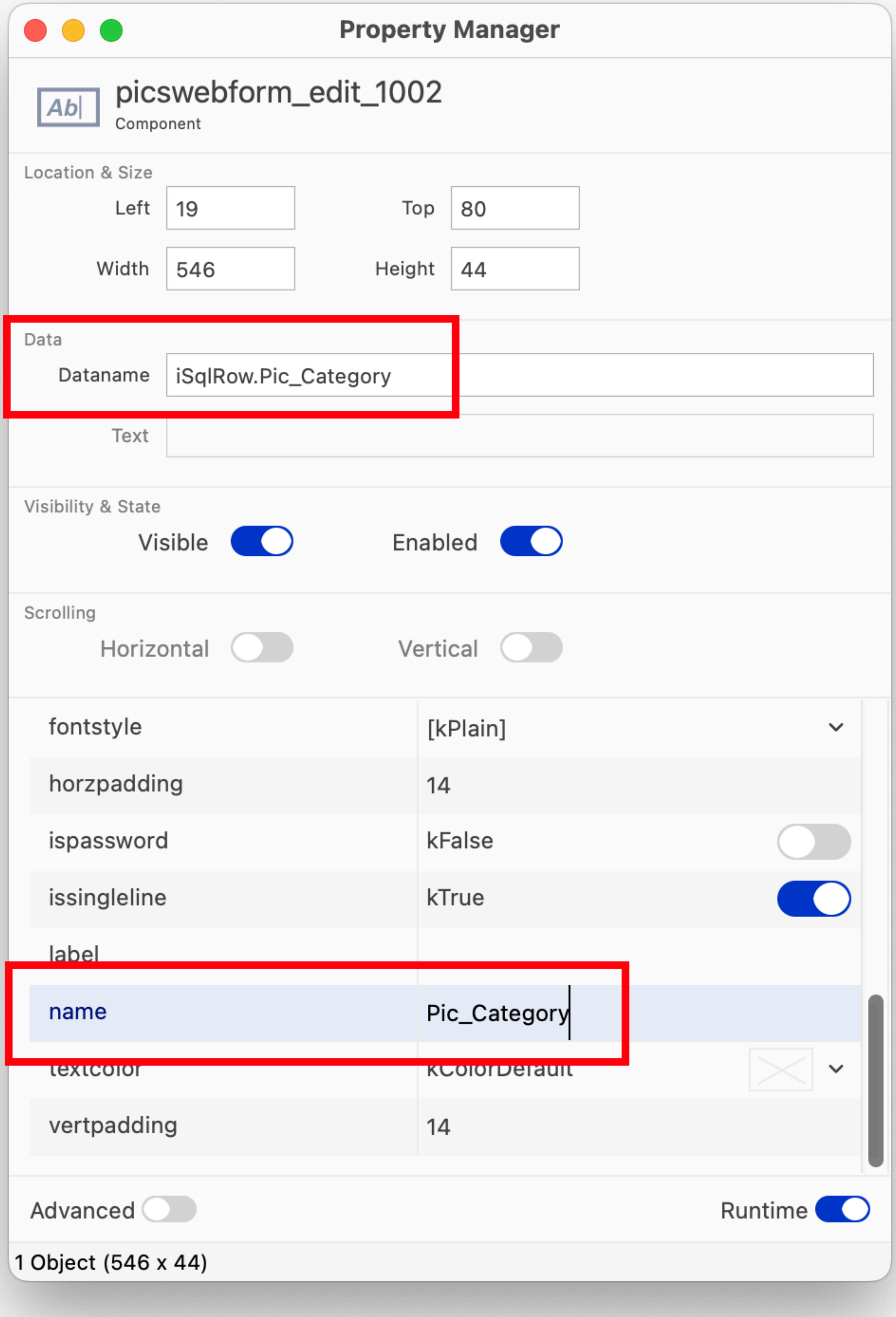
Adding a Picture and a Button to a Form

Next you need to add a **Picture** control to your form. You can find the Picture control in the Media group in the Component Store, which should be docked to the left of the Remote form editor.

- Locate the **Picture** control in the **Media** group in the Component Store

- Drag the **Picture** control from the Component Store and drop it onto your remote form, placing it under the Pic_Name field.
- Resize the **Picture** control by dragging its handles, and use the Position Assistance to line up the left edge with the Pic_Name field, and its top edge with the Pic_Desc field.

- *Make sure the Picture control is selected,* and press **F6/Cmnd-6** to open the Property Manager.



Property Manager

picswebform_edit_1002
Component

Location & Size

Left	19	Top	80
Width	546	Height	44

Data

Dataname: iSqlRow.Pic_Category

Text:

Visibility & State

Visible: on — Enabled: on

Scrolling

Horizontal: off — Vertical: off

fontstyle	[kPlain]
horzpadding	14
ispassword	kFalse
issingleline	kTrue
label	
name	Pic_Category
textcolor	kColorDefault
vertpadding	14

Advanced: off — Runtime: on

1 Object (546 x 44)

Figure 47:

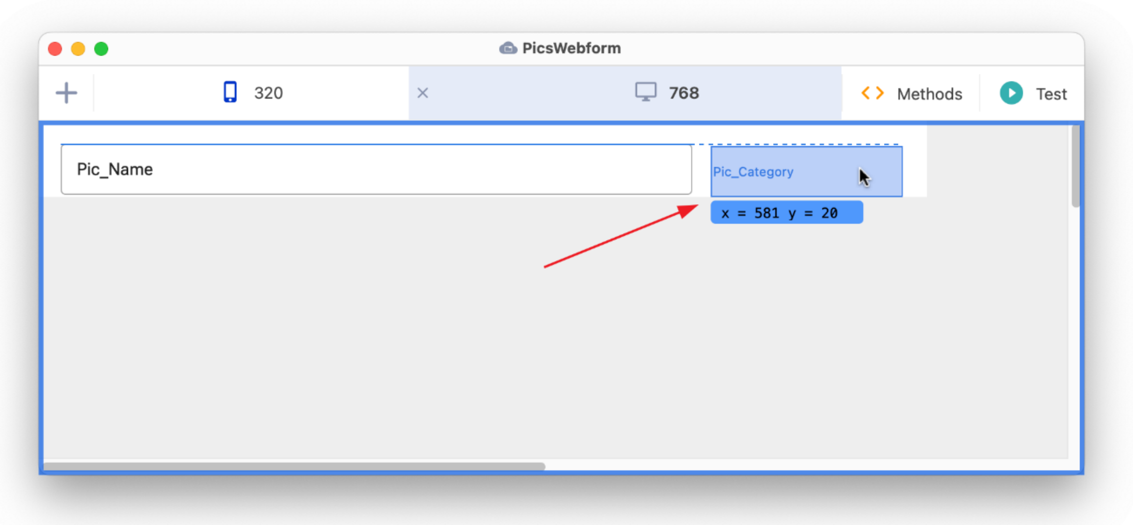


Figure 48:

- Change the properties as follows (you can type the variable name iSqlRow or select it from the Code Assistant, then add a dot (fullstop/period) and add the column name Pic_URL):

  dataname = iSqlRow.Pic_URL
  name = Pic_Image

The Pic_URL column contains the location (relative path) of each picture referenced in the SQLite database.

Next you need to create a **Button** in your form to allow the end user to load each data record, that is, each row in the database.

- Locate the **Button** control in the Buttons group in the Component Store

- Drag the Button onto your form and drop it under the Picture control.
- *With the button selected,* press **F6/Cmnd-6** to open the Property Manager (or bring it to the front), and change its properties as follows:

  name = Next
  text = Next

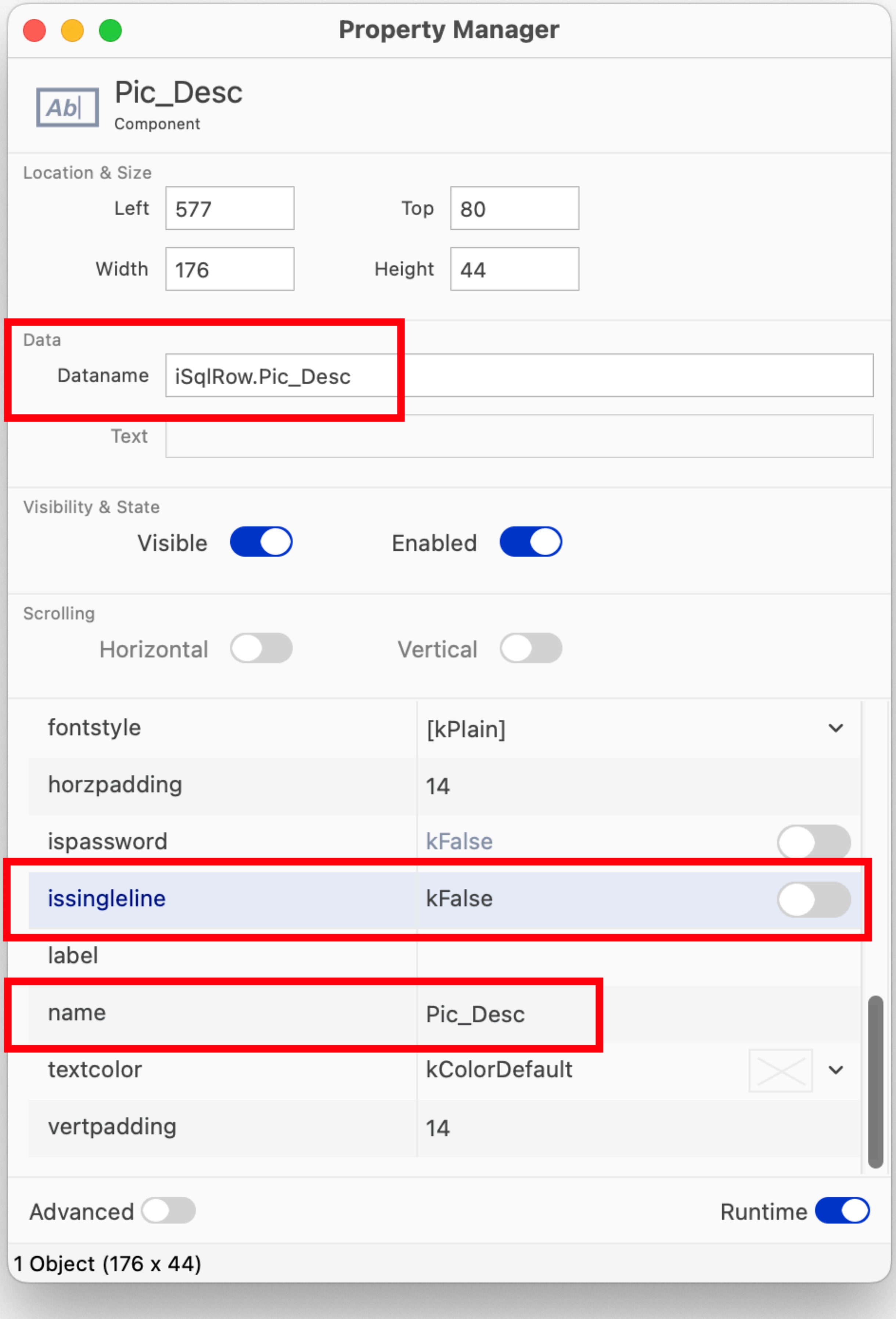
You need to add some code to the **Next** button; you can copy the code text from here:

```
On evClick
Do iSqlRow.$fetch() Returns lStatus
If lStatus=kFetchFinished=kFetchError
  Do iSqlRow.$select()
  Do iSqlRow.$fetch() Returns lStatus
End If
Calculate iOldRow as iSqlRow
```

(Alternatively, if you are using the Professional Edition or free trial, you can copy the code from the Next button on the PicsWindow window class you created previously in this tutorial, or from one of the **pics** libraries in the 'final' folder; you'll need to copy the code from the $event method for the Next button.)

- ***Double-click*** on the **Next** button in the **PicsWebform** remote form to open the Method Editor for the button.

Note when you double-click on an object, Omnis will open the Method Editor and select the object's **$event** method, in this case, $event for the **Next** button on your form. This method will contain the code that will be executed *when the end user clicks on the button*, i.e. it is the event method for the button.



Property Manager

Pic_Desc
Component

Location & Size

Left	577	Top	80
Width	176	Height	44

Data

Dataname: iSqlRow.Pic_Desc

Text:

Visibility & State

Visible ● Enabled ●

Scrolling

Horizontal ○ Vertical ○

fontstyle	[kPlain]
horzpadding	14
ispassword	kFalse
issingleline	kFalse
label	
name	Pic_Desc
textcolor	kColorDefault
vertpadding	14

Advanced ○ Runtime ●

1 Object (176 x 44)

Figure 49:

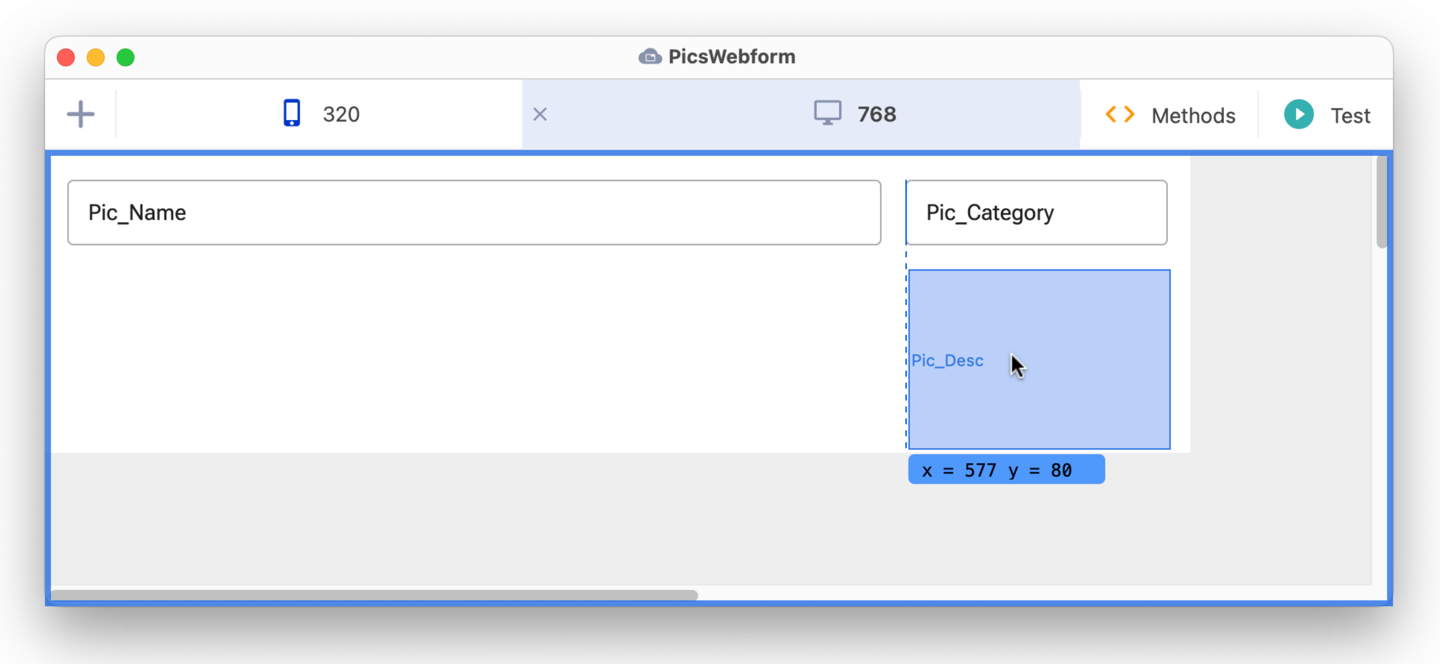


Figure 50:

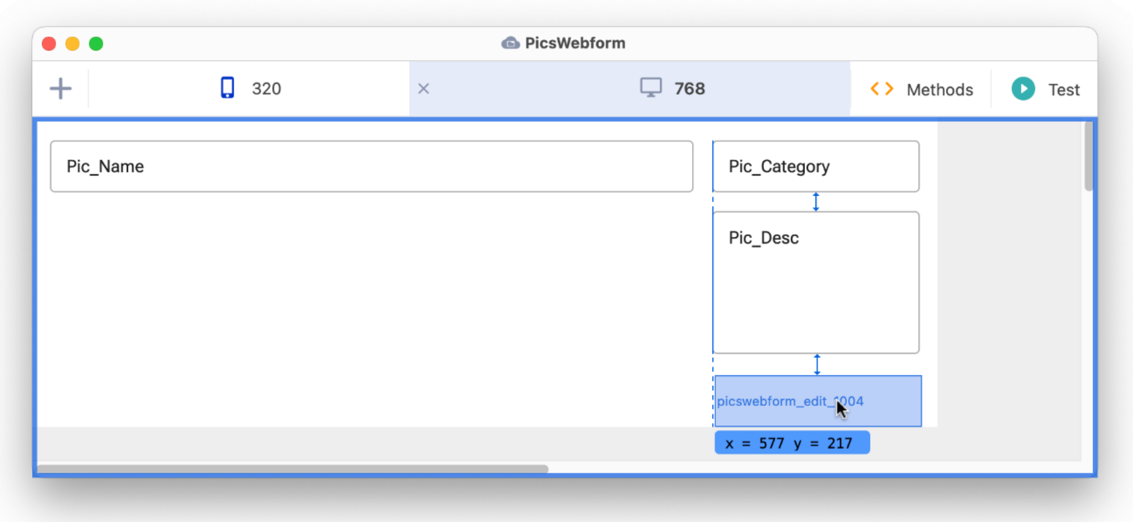


Figure 51:

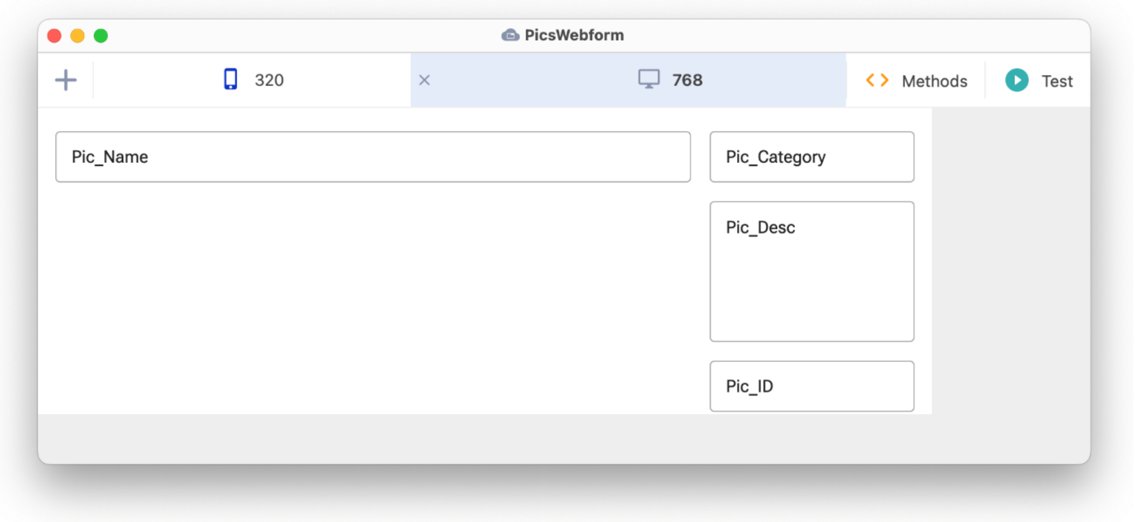


Figure 52:

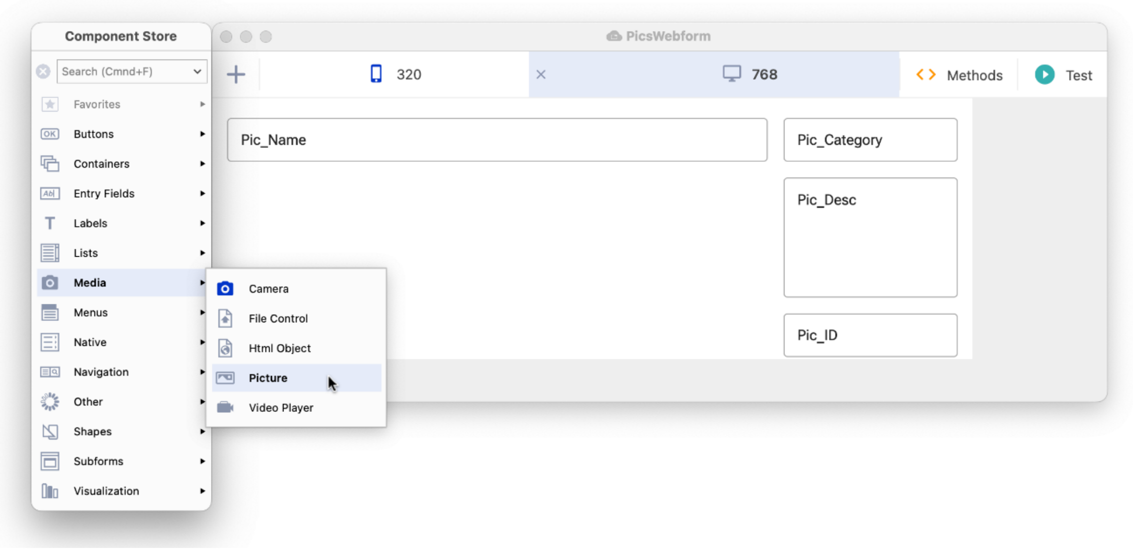


Figure 53:

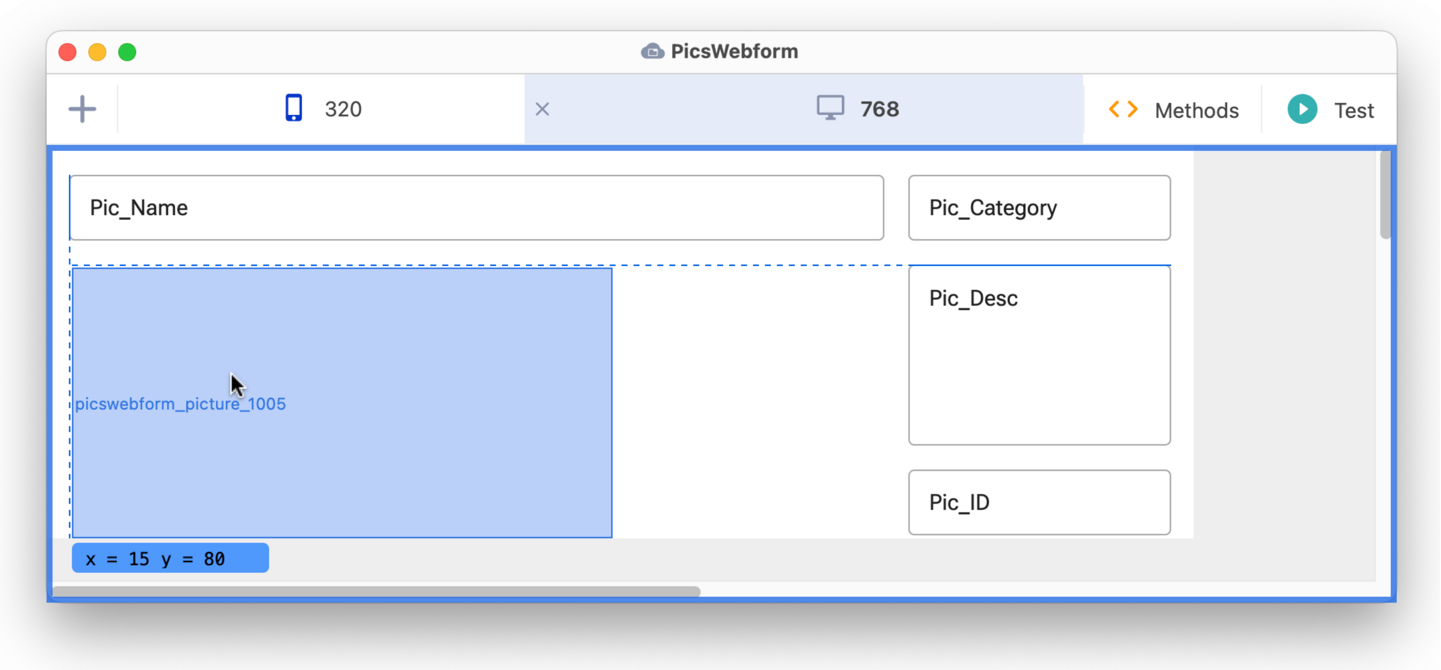


Figure 54:

- Select the whole of the first line of code ***including On evClick*** (click in the code line and press Ctrl/Cmnd-A to select all) and *paste the code from the clipboard* replacing the whole of the existing line.

Note that if you copied the code from this page you will need to create the *local* variable **lStatus** type = *Character,* and the *instance* variable **iOldRow** type = *Row.* (If you copied the code from the Next button in the PicsWindow class, the variables will also have been copied automatically, so you can skip the next part about creating the variables.)

To create the variables, click in the variable name in the Code Editor, click on the Fix button (at the bottom of the Code editor window), and create the variable in the **Create Variable** dialog, remembering to select the correct Scope and Type for each (although Omnis will try to select these for you automatically).

This is the definition for **lStatus:**

This is the definition for **iOldRow:**

The iOldRow and iSqlRow variables are needed to load or "fetch" each successive row of data from the database (in the result set from the database SELECT).

Your Method Editor should look the same as the following, including the syntax color coding for the variables you defined:

(If you copied the code from the PicsWindow window class you can delete the last line of code 'Do $cwind.$redraw' since it will have no effect in a web form, or you can comment it out by clicking anywhere in the line (or selecting the whole) and pressing Ctrl/Cmnd-/. A # symbol is added to the line and it becomes inactive code.)

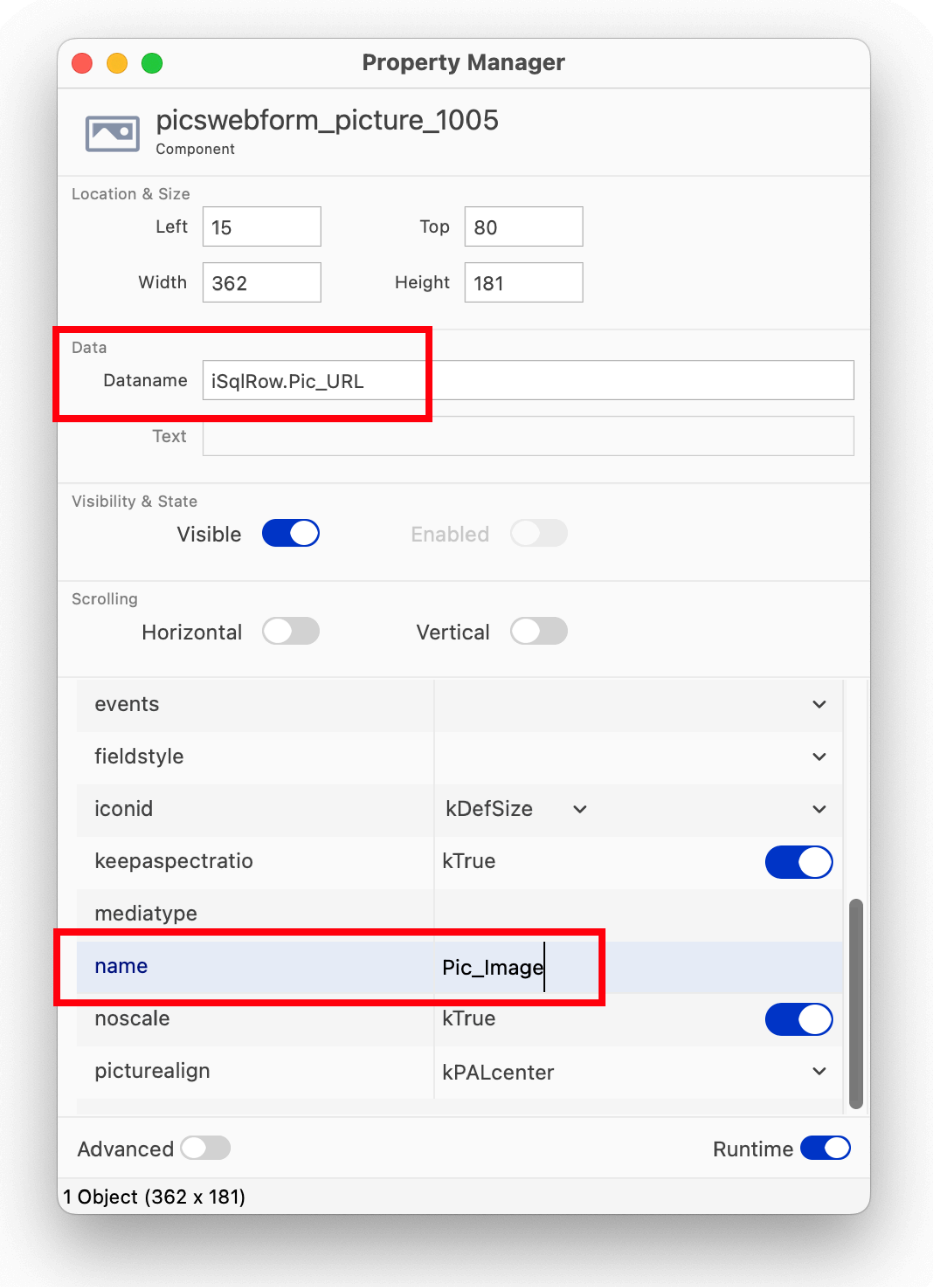
- Close the Method Editor for the PicsWebform.

Adjusting the form for a Mobile Device

At present the fields on your PicsWebform are positioned to be displayed in a browser on a desktop computer or tablet (on the 768 layout breakpoint), but you can reposition the fields on the 320 breakpoint so the same remote form can be displayed on different devices, including mobile phones.

- Open the **PicsWebform** in design mode, or click on it to bring it to the top.
- Click on **320** in the Design bar at the top of the Remote form editor to select the 320 layout breakpoint.

To tidy up the fields on the 320 breakpoint layout you can start with the layout from the 768 breakpoint you already designed (at this stage the fields are randomly placed).



Property Manager

picswebform_picture_1005
Component

Location & Size

Left 15
Top 80
Width 362
Height 181

Data

Dataname iSqlRow.Pic_URL
Text

Visibility & State

Visible: on
Enabled: off

Scrolling

Horizontal: off
Vertical: off

events	
fieldstyle	
iconid	kDefSize
keepaspectratio	kTrue
mediatype	
name	Pic_Image
noscale	kTrue
picturealign	kPALcenter

Advanced: off
Runtime: on

1 Object (362 x 181)

Figure 55:

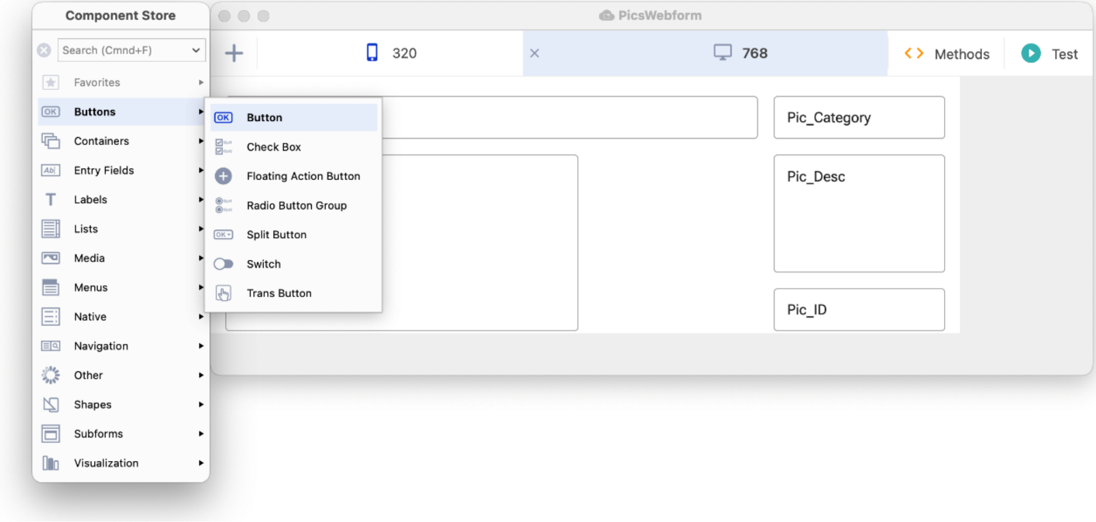
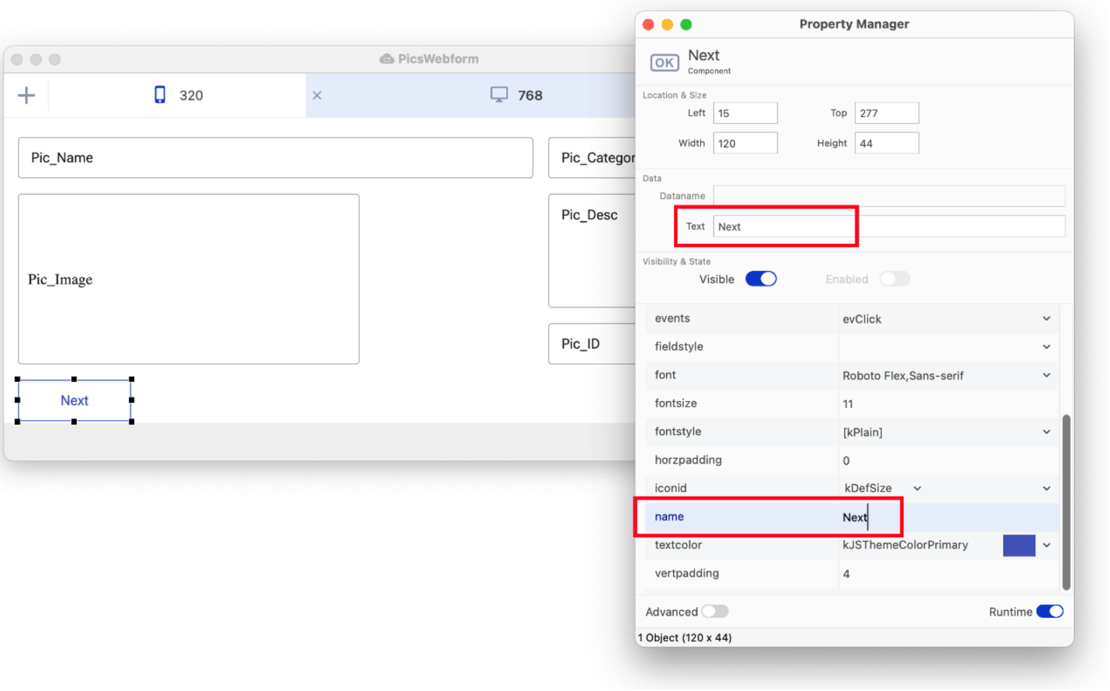


Figure 56:



Next

Location & Size

Left 15 Top 277

Width 120 Height 44

Data

Dataname

Text Next

Visibility & State

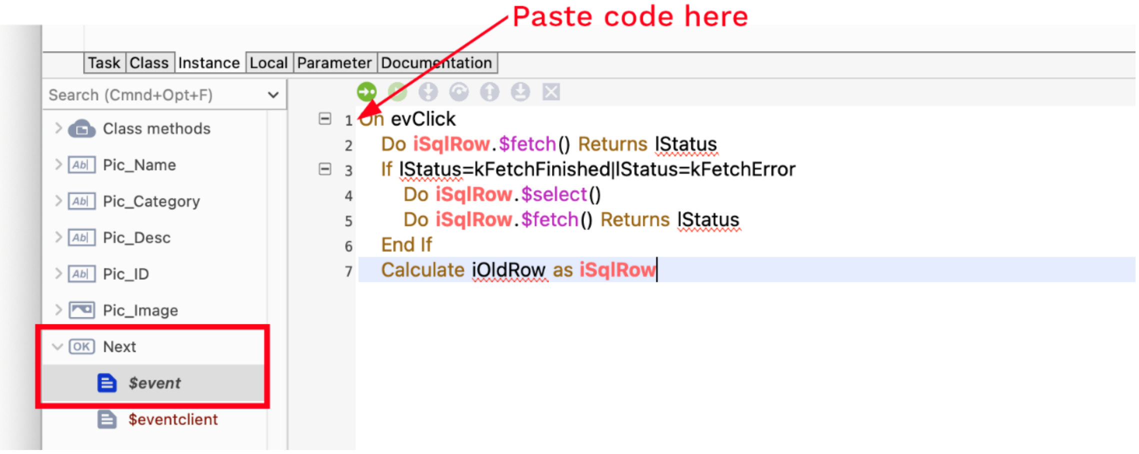
Visible ● Enabled ○

events	evClick
fieldstyle	
font	Roboto Flex,Sans-serif
fontsize	11
fontstyle	[kPlain]
horzpadding	0
iconid	kDefSize
name	Next
textcolor	kJSThemeColorPrimary
vertpadding	4

Advanced ○ Runtime ●

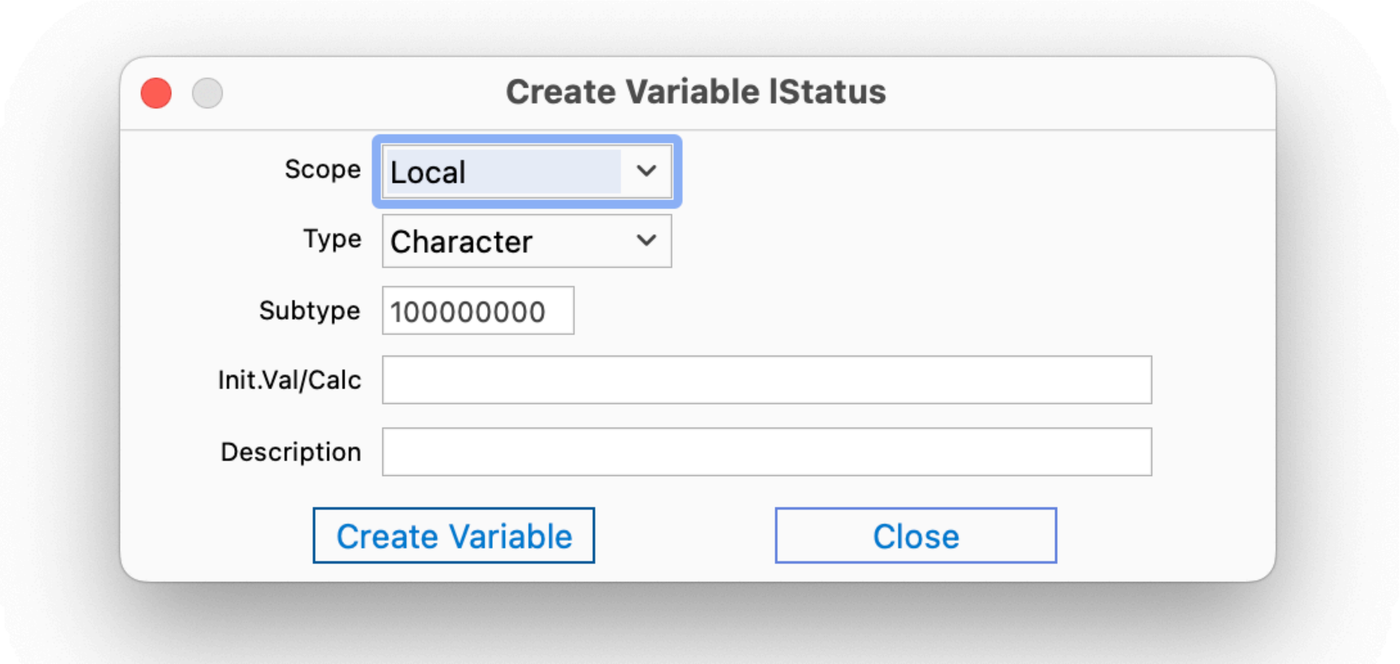
1 Object (120 x 44)

Figure 57:



```
On evClick
  Do iSqlRow.$fetch() Returns lStatus
  If lStatus=kFetchFinished|lStatus=kFetchError
    Do iSqlRow.$select()
    Do iSqlRow.$fetch() Returns lStatus
  End If
  Calculate iOldRow as iSqlRow
```

Figure 58:



Create Variable lStatus

Scope: Local

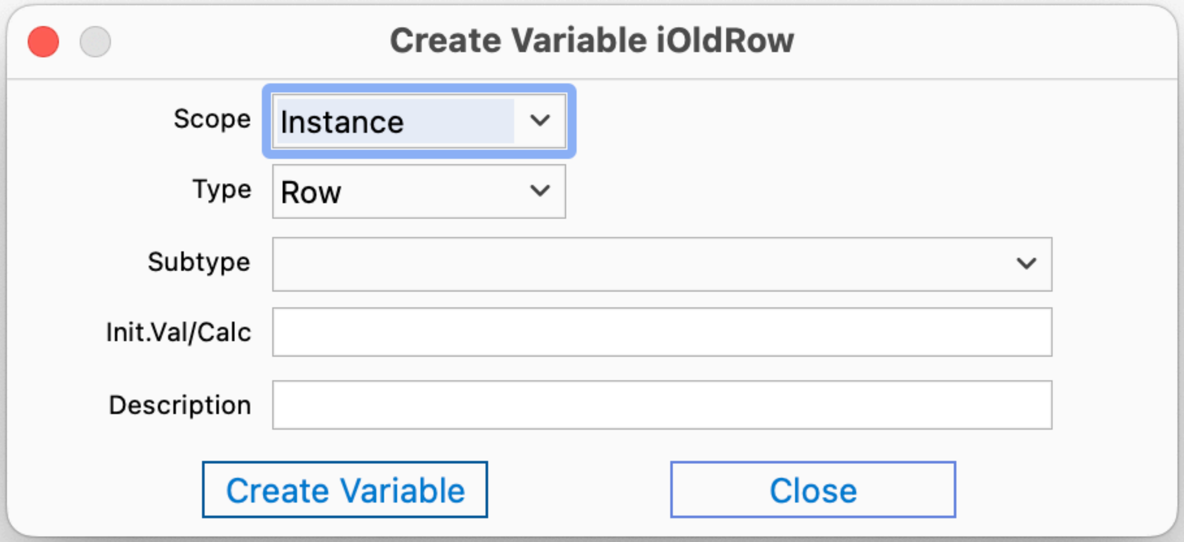
Type: Character

Subtype: 100000000

Init.Val/Calc:

Description:

Figure 59:



Create Variable iOldRow

Scope: Instance

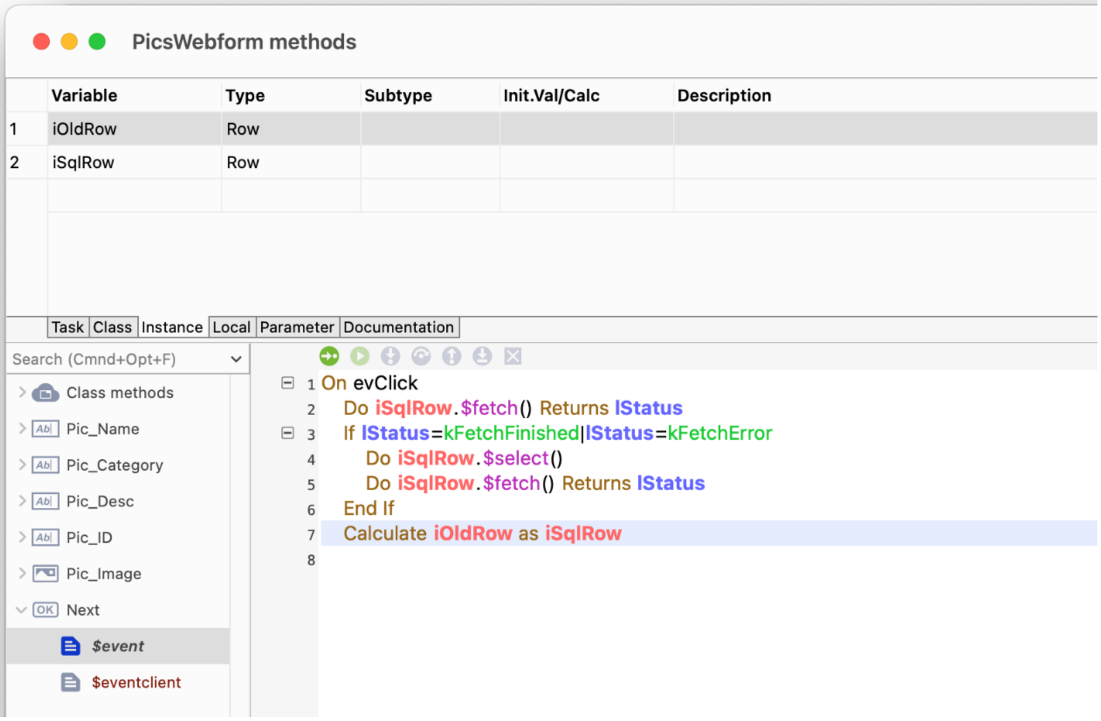
Type: Row

Subtype:

Init.Val/Calc:

Description:

Figure 60:



PicsWebform methods

	Variable	Type	Subtype	Init.Val/Calc	Description
1	iOldRow	Row			
2	iSqlRow	Row			

```
On evClick
  Do iSqlRow.$fetch() Returns lStatus
  If lStatus=kFetchFinished|lStatus=kFetchError
    Do iSqlRow.$select()
    Do iSqlRow.$fetch() Returns lStatus
  End If
  Calculate iOldRow as iSqlRow
```

Figure 61:

- Right-click on the background of the form and select the **Copy Layout from Breakpoint** option and select the **768** option

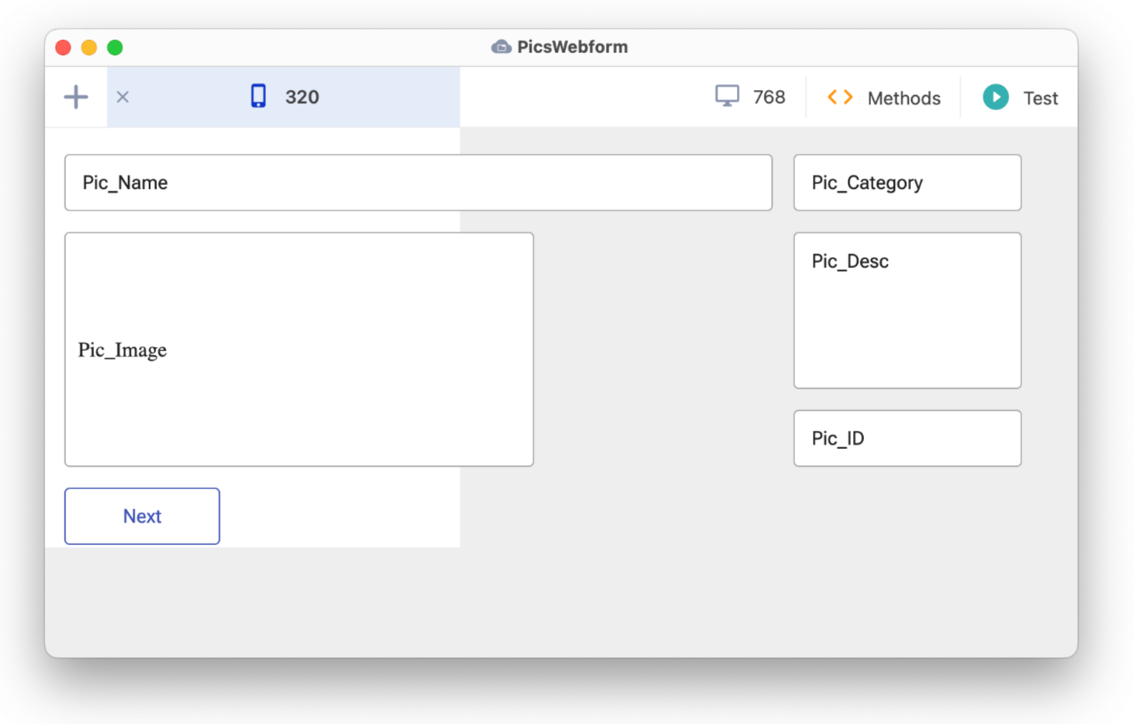


Figure 62:

- Drag the lower edge or corner of the remote form to make the form larger, that is, increase its height.
- Drag the Next button down (or position it at the top if you would prefer), then resize and re-position all the other fields to fit the width of the 320 layout breakpoint; you can use the Position Assistance to help you line up the controls (as shown below).

Your remote form should look something like the following:

To test your remote form in a web browser, the library needs a Remote Task, which handles the client connections at runtime when you test or deploy your application. However, when you test a remote form, and if your library does not contain a Remote task, one will be created for you automatically and assigned to your Remote form.

So you can skip to the Testing your Web Form section.

Creating a Web Form using a wizard

If you have created a web form from scratch, described in the previous section, you can skip this section; go to the Testing your Web Form section.

Having created the database session and data structure (the MyPictures schema), you can create a **Remote Form** to browse the data in a web browser on your desktop computer or on a mobile device. The previous section described how to create a Remote form from scratch (and adding fields manually), but this section describes how you can create a Remote form using a ***Remote Form Wizard;*** if you are evaluating Omnis Studio then using a wizard will speed up the development process allowing you to assess Omnis in a shorter time.

Remember, if you have quit and restarted Omnis Studio, then you will need to reopen your database session, described in the 'Opening a Database Session' section, to complete this section (if the PicSess session is open it will be listed in the SQL Browser as highlighted below).

- Click the **Pics** library in the Folders tree list in the Studio Browser and select the **Class Wizard** option.

- Now click on the **Remote Form...** option in the Browser list to display the available remote form class wizards (some of these options will not appear in the Community edition).

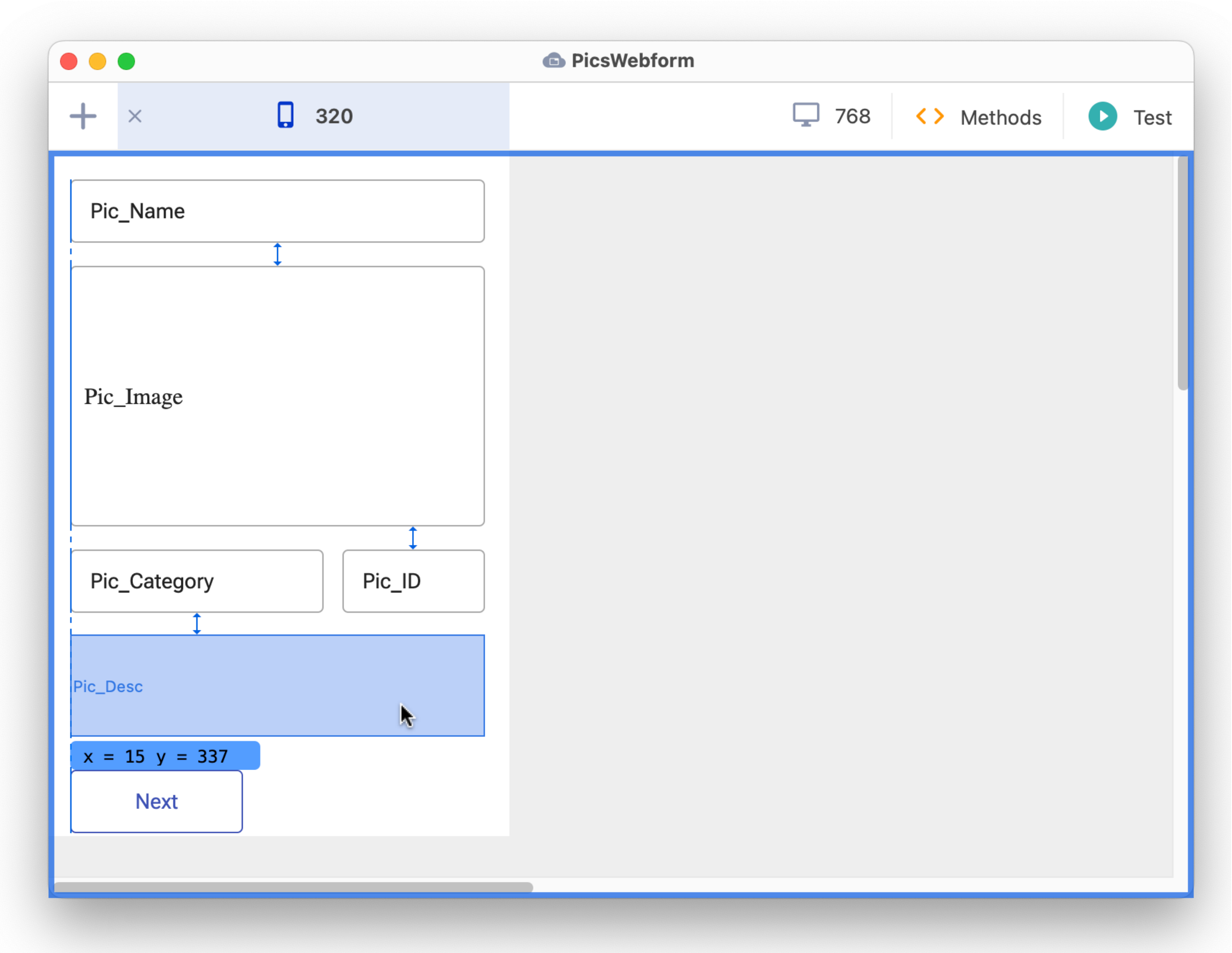


Figure 63:

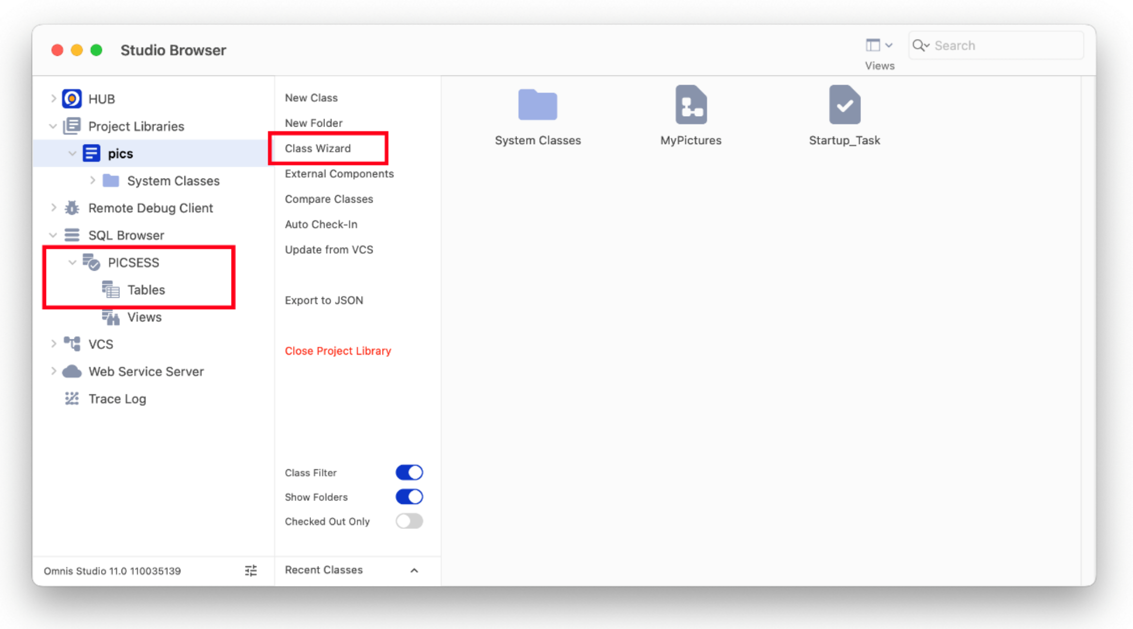


Figure 64:

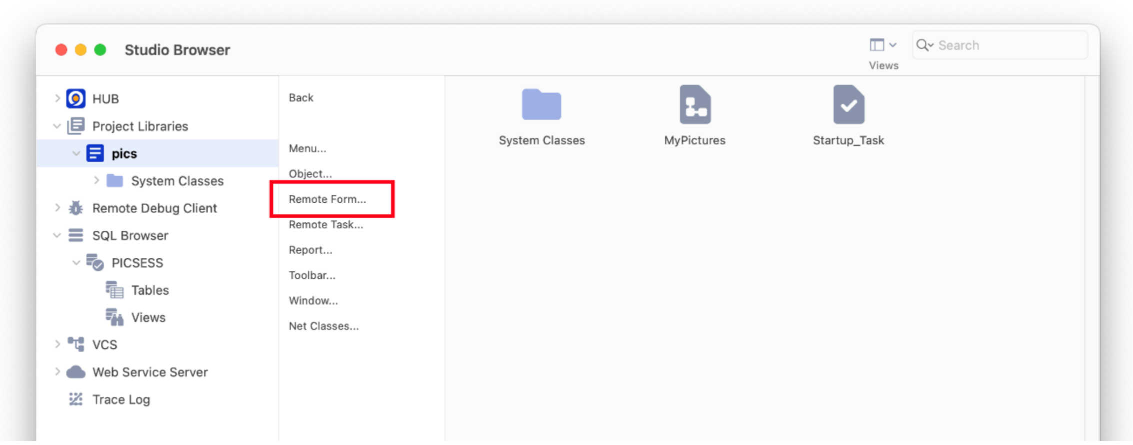
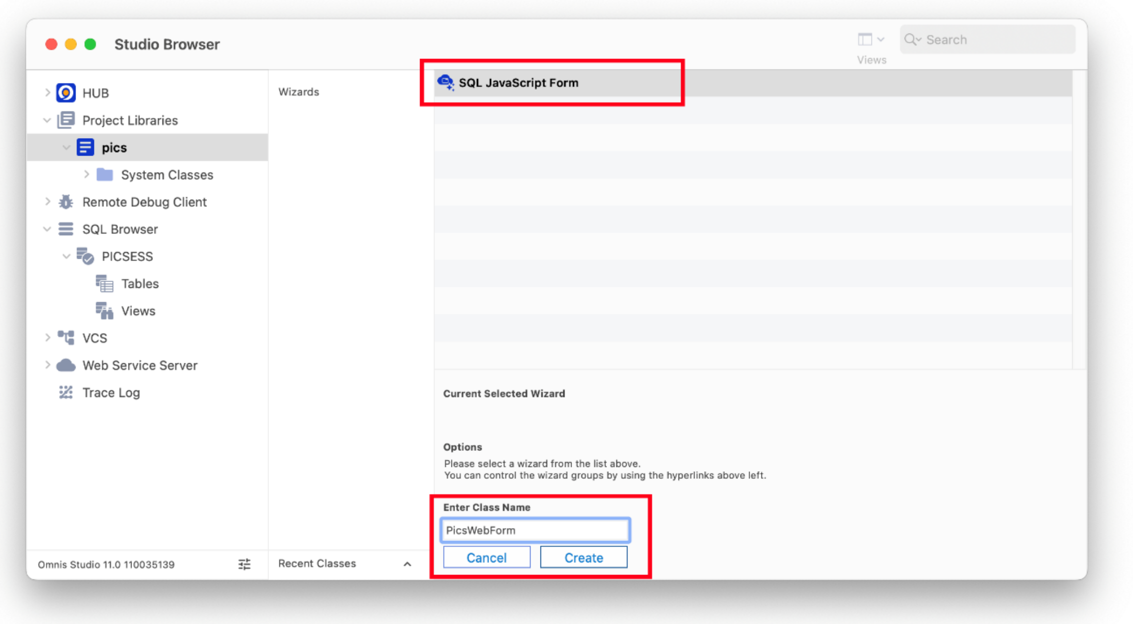


Figure 65:

- Select the **SQL JavaScript Form** option in the list, enter the class name **PicsWebForm** and press the **Create** button.



Current Selected Wizard

Options
Please select a wizard from the list above.
You can control the wizard groups by using the hyperlinks above left.

Enter Class Name: PicsWebForm

Figure 66:

Next you need to select a Remote task, and since you are creating a Remote form and your library does not contain a Remote task, Omnis will create one for you automatically.

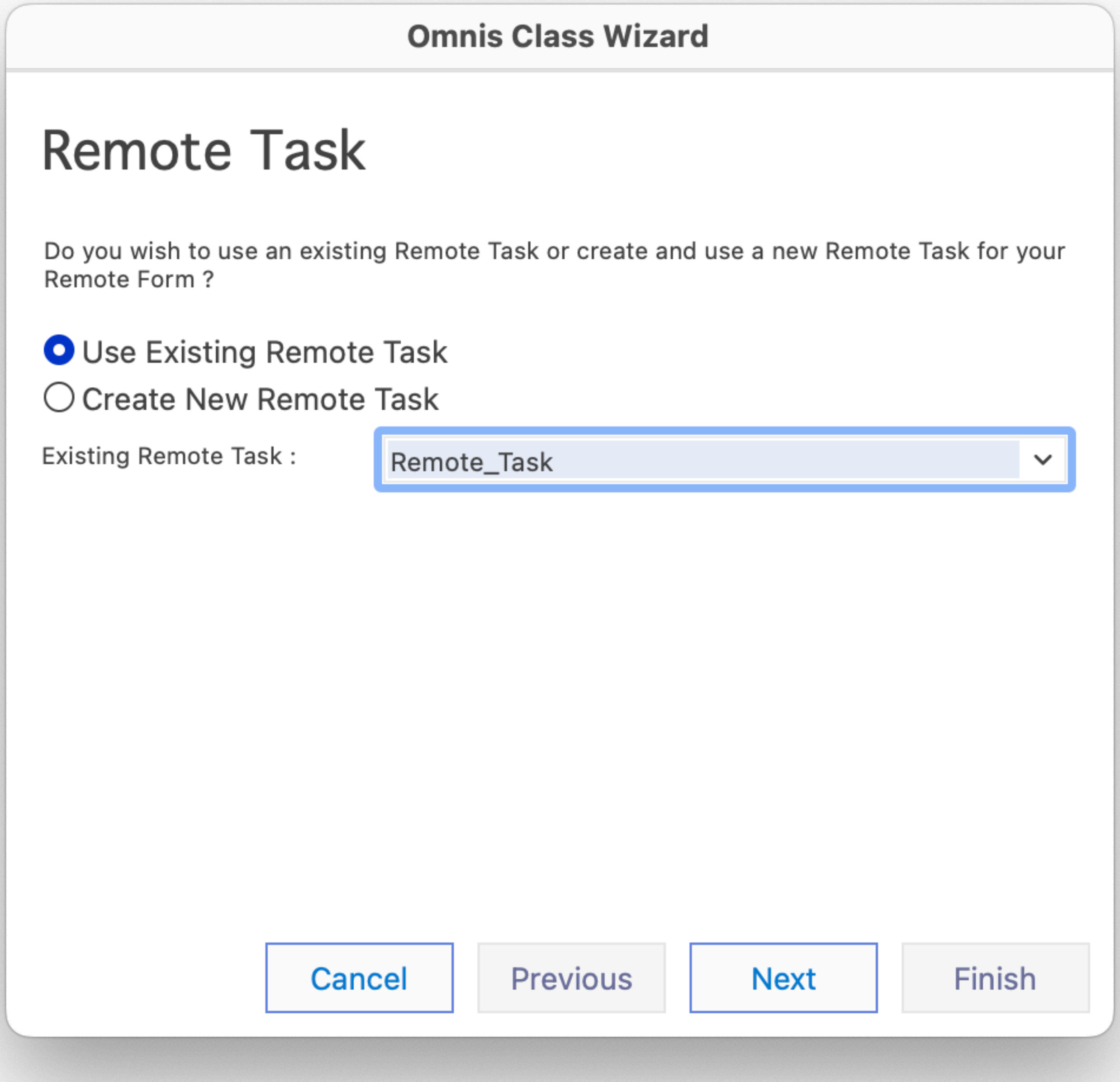
- Select the **Use Existing Remote Task** option (the default option).
- Select the **Remote_Task** (this will be created for you automatically).

- Click on **Next** after selecting a remote task

Next in the Class wizard screen, you need to choose the style or layout of the remote form; the radio buttons allow you to select the style of remote form to create.

The *"One field per column based on schema or query class"* option should be selected by default, which creates a simple form to allow you to browse the data in a single database table, like the one used in this tutorial.

If you have more time, and want to explore Omnis Studio a little more, you can come back to this stage of the tutorial and try the other form layouts, such as the Display or Enterable grid-based form; note the Parent / Child option requires two schema classes linked in a parent/child relationship.



Omnis Class Wizard

Remote Task

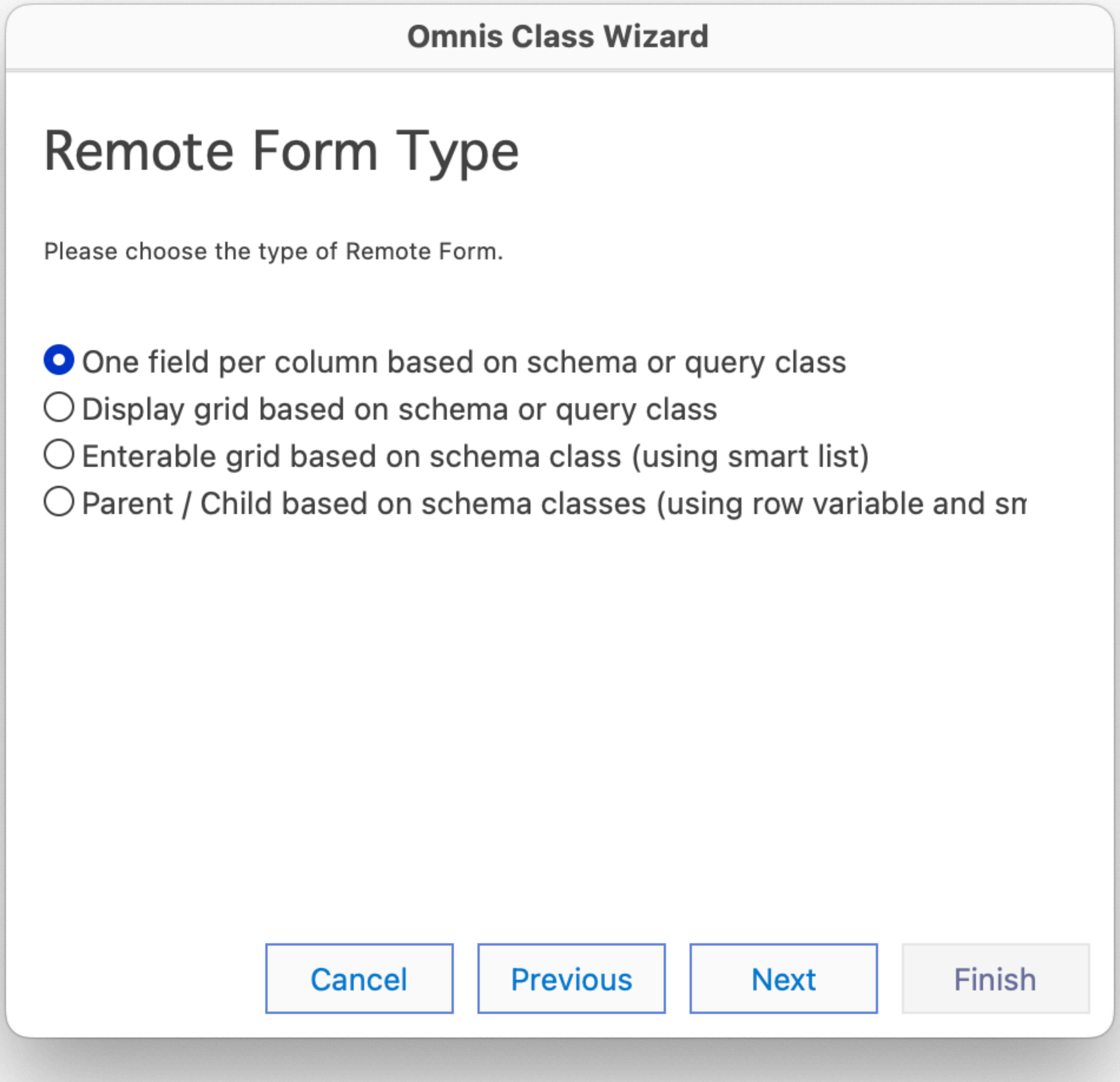
Do you wish to use an existing Remote Task or create and use a new Remote Task for your Remote Form ?

● Use Existing Remote Task

○ Create New Remote Task

Existing Remote Task : Remote_Task

Figure 67:



Omnis Class Wizard

Remote Form Type

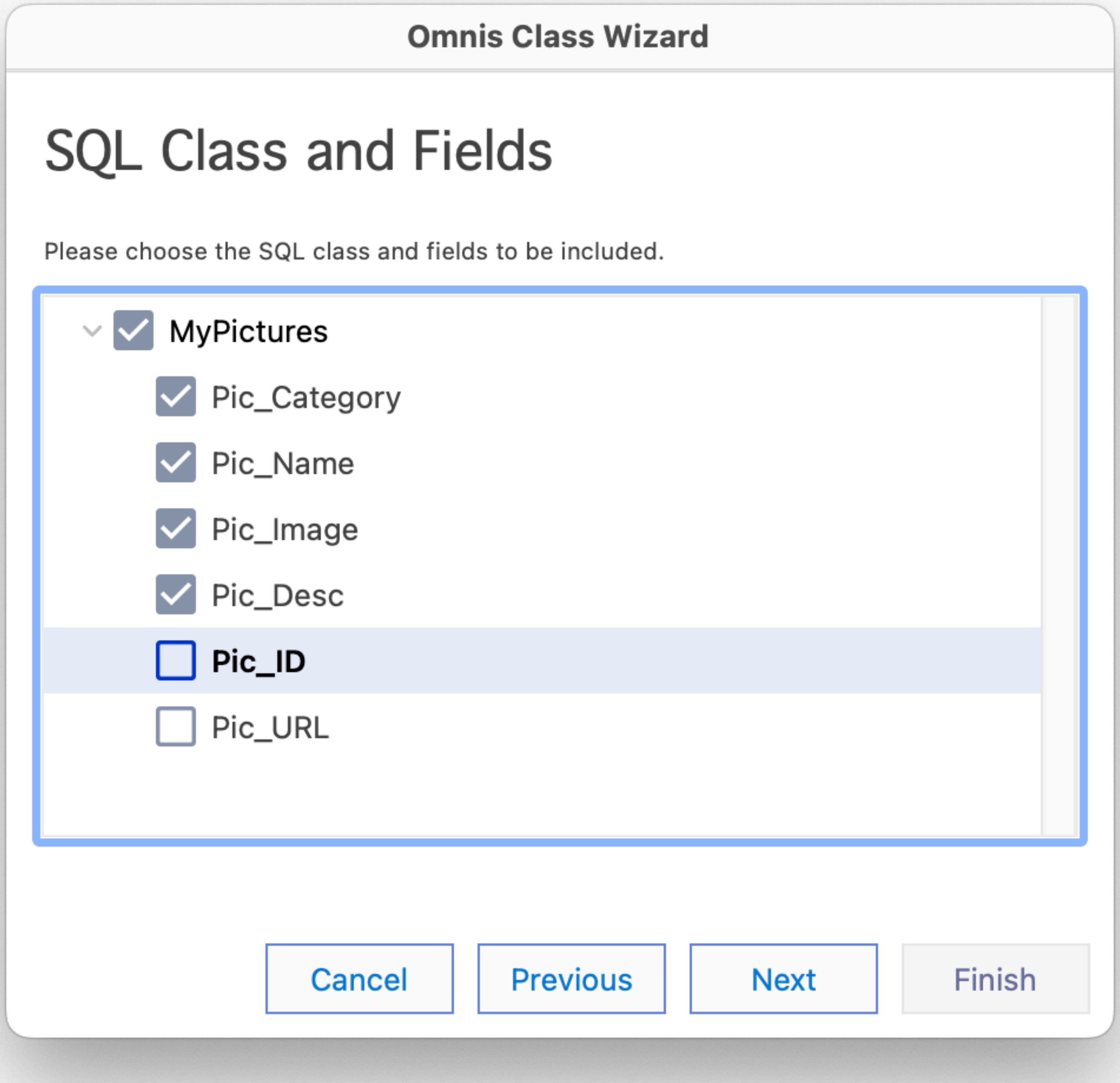
Please choose the type of Remote Form.

● One field per column based on schema or query class
○ Display grid based on schema or query class
○ Enterable grid based on schema class (using smart list)
○ Parent / Child based on schema classes (using row variable and sn

Figure 68:

- Make sure the *"One field per column based on schema or query class"* option is selected and click on **Next.**

Note that at any time in the wizard process you can press the **Previous** button to go back and review or change a selection. Next you need to select the SQL class and fields you want to include in your remote form.



Omnis Class Wizard

SQL Class and Fields

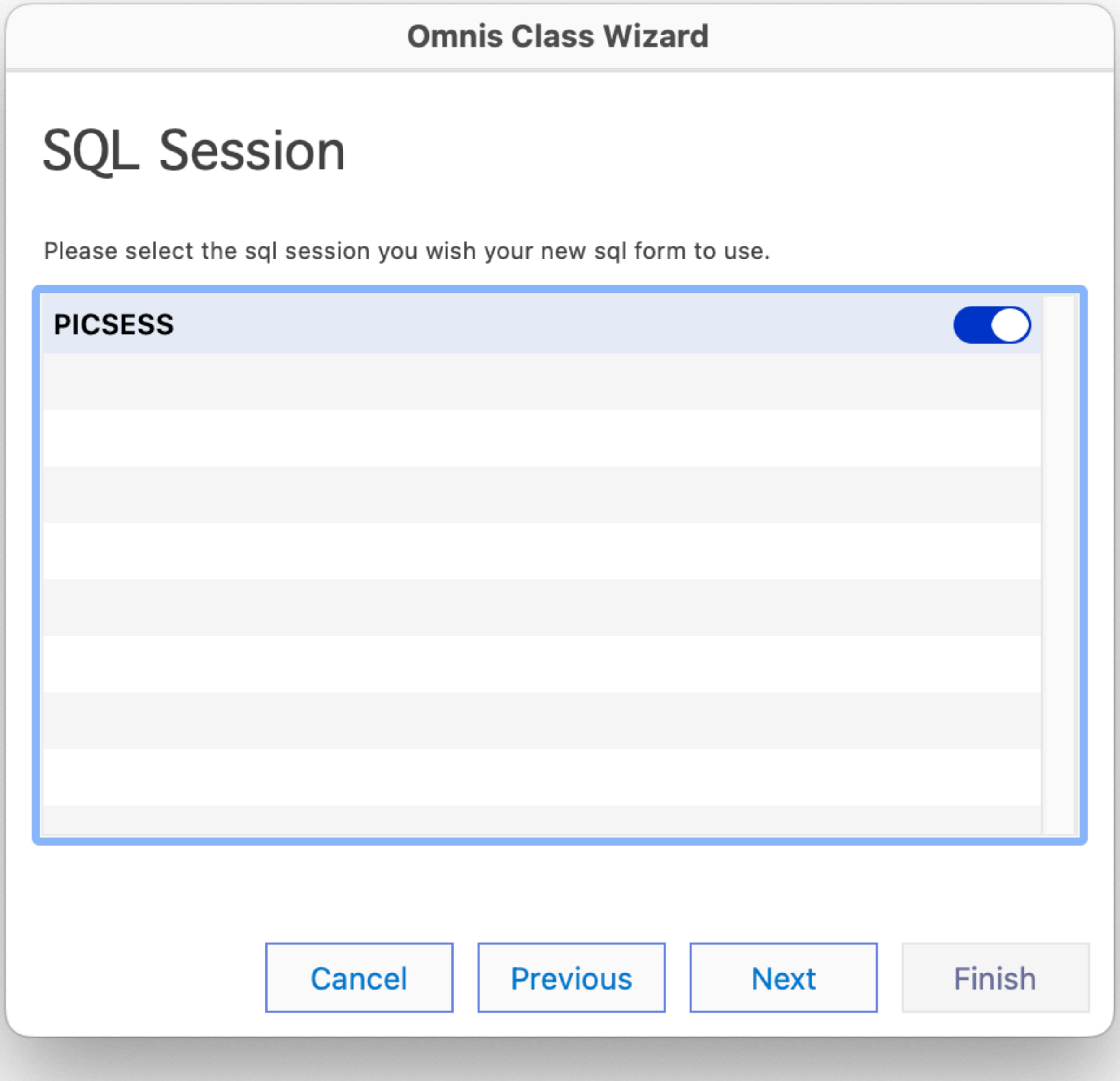
Please choose the SQL class and fields to be included.

☑ MyPictures
- ☑ Pic_Category
- ☑ Pic_Name
- ☑ Pic_Image
- ☑ Pic_Desc
- ☐ Pic_ID
- ☐ Pic_URL

Figure 69:

- Click on the check box to select the **MyPictures** schema, then click on the arrow (>) icon to view the fields (columns) in the schema and ***deselect*** the Pic_ID and Pic_URL options since we do not need to display them on the remote form (this step is not essential but simplifies your form).
- Press the **Next** button to move on to the stage where you will choose the database connection that will be used to access the data.

Note that if at this point you get the message *"There are no SQL sessions available for selection. Please use the SQL Object Browser to open a session and try again*", you need to cancel the wizard and open the PicSess database connection, as described earlier in the 'Opening a Database Session' section, and then restart this section.



Omnis Class Wizard

SQL Session

Please select the sql session you wish your new sql form to use.

PICSESS

Figure 70:

- Enable the switch to select the **PICSESS** database session and press the **Next** button to move on to the final stage.
- Press the **Finish** button to create the remote form.

The new JavaScript Remote form is created for you and opened in design mode; the **768 layout breakpoint** is selected and the Component Store is docked to the left side of the Remote form editor. A Remote_Task has also been created.

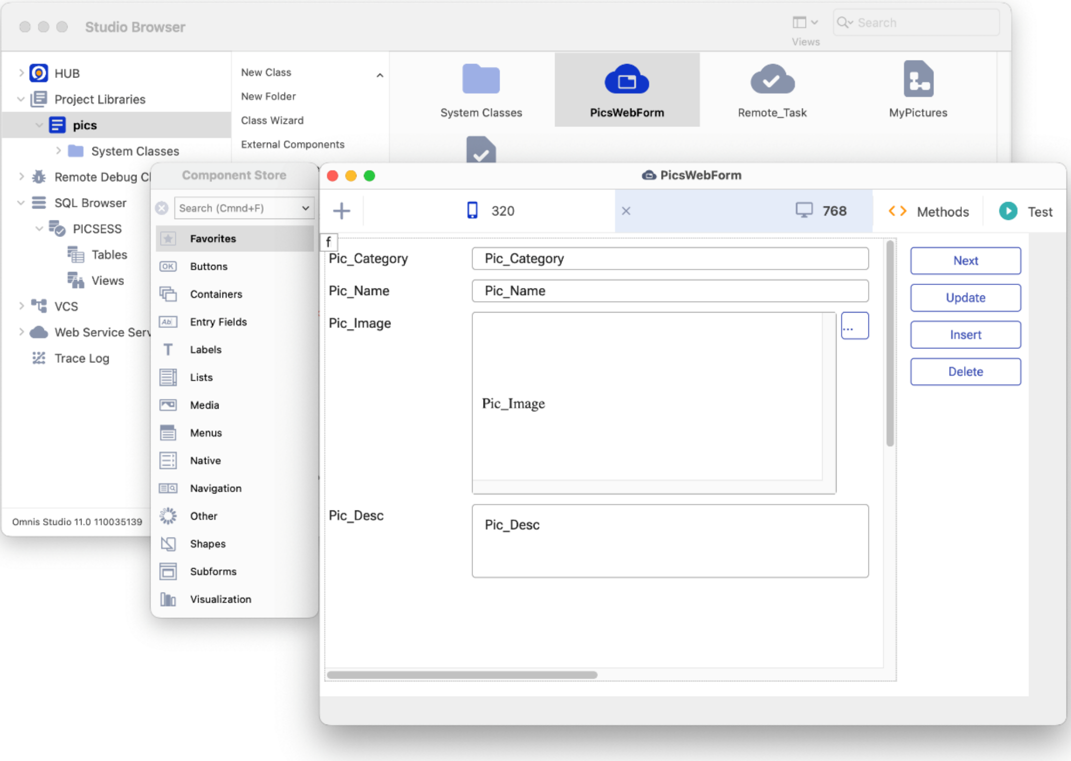


Figure 71:

Omnis has added all the required fields and buttons, and created 2 form layouts (shown in the Design bar at the top) to cater to *desktop browsers* (768 breakpoint) and *mobile devices* (320 breakpoint); click on each layout to see how the controls have been arranged.

You can click on the Methods button in the Design bar to examine the code Omnis has added automatically during the wizard process; close the Method Editor window(s) when you've finished. Omnis has added an Upload button (shown with ellipsis, to the right of the Pic_Image control) and code to allow you to upload an image; when you test the form you can enter a new record including the PNG image provided in the zip (this is described later).

Adding some code

Before you can open the remote form and use it to insert data, you must make a small change to the **Insert** button. The **Pic_ID** column in the SQLite database you are using is an INTEGER data type with a *Primary Key*, known as an INTEGER PRIMARY KEY, which will store a unique integer value automatically. The "trick" is to insert the value of Pic_ID as NULL and SQLite will insert the next available numeric value; in effect the value of Pic_ID is incremented by 1 automatically and the new value is inserted. If you delete a data record, the value of Pic_ID in that record is never reused; SQLite will always use a new unique value when inserting a new row.

- Assuming you have the **PicsWebForm** open in design mode, ***double-click*** on the **Insert** button, or Right-click on the button and select **Field Methods...** (or select the button and press Shift-F8) to display the methods for the fields or controls on the form.

The Omnis **Method Editor** will open and the **$event method** behind the Insert button will be highlighted. This method will be executed when the end user clicks on the button; in this case the method inserts the current values in the iSqlRow variable into the database, and it's before this line you need to make a small addition.

- Click anywhere in the second line of code 'Do iSqlRow.$insert() Returns lReturnFlag' and press **Ctrl-I/Cmnd-I** to *insert an empty line above*.

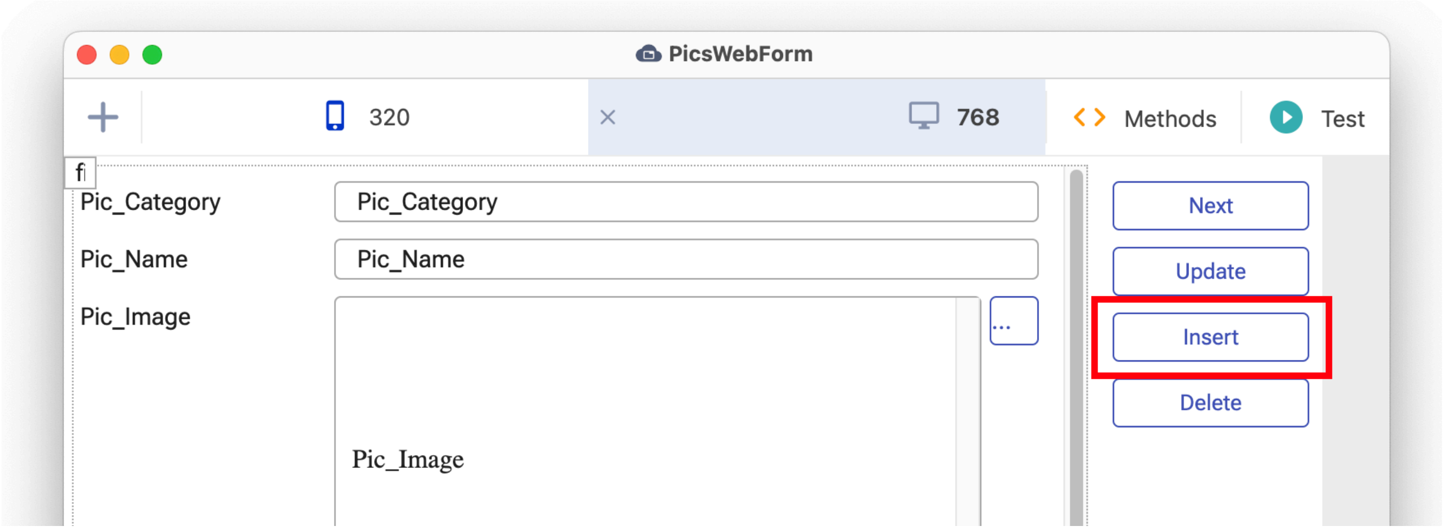
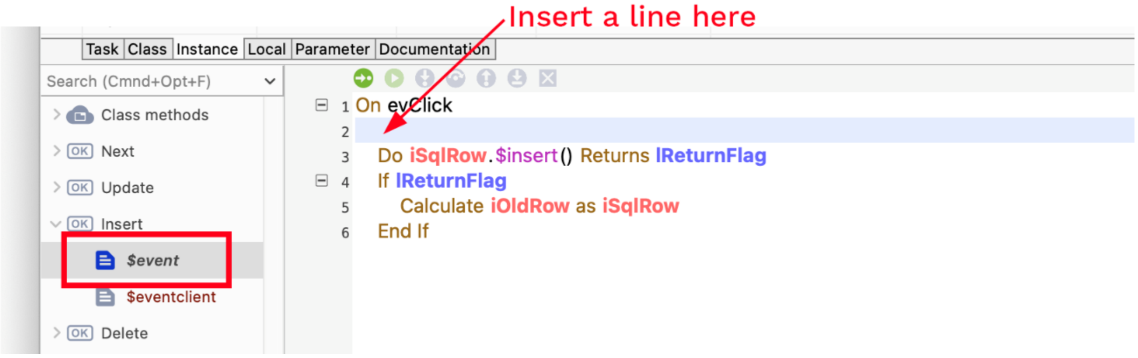


Figure 72:

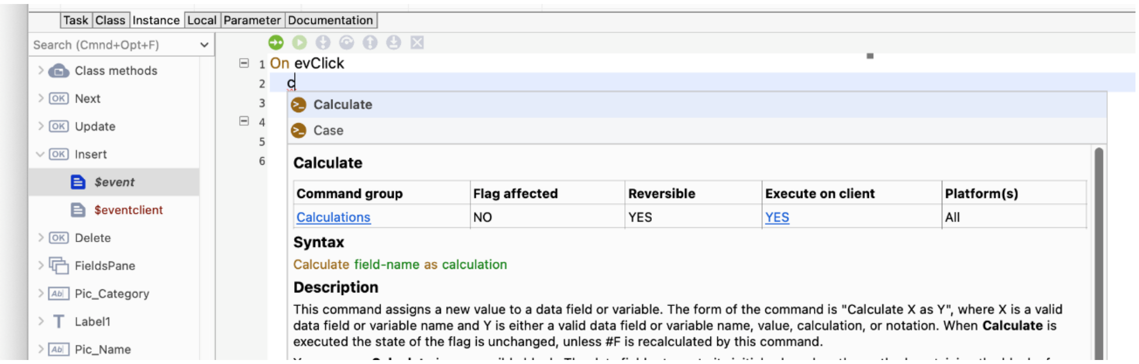


Insert a line here

```
On evClick

 Do iSqlRow.$insert() Returns lReturnFlag
 If lReturnFlag
    Calculate iOldRow as iSqlRow
 End If
```

Figure 73:

- With the cursor on the empty line, press "c" to locate the *Calculate* command: the Code Assistant will drop down from the code line and display a list of commands starting with the letter "C". In this case, the Calculate command is the first command in the list so, *assuming Calculate is selected,* press the **Return** key to enter the command.



```
On evClick
    c
```

Calculate

Command group	Flag affected	Reversible	Execute on client	Platform(s)
Calculations	NO	YES	YES	All

Syntax

Calculate field-name as calculation

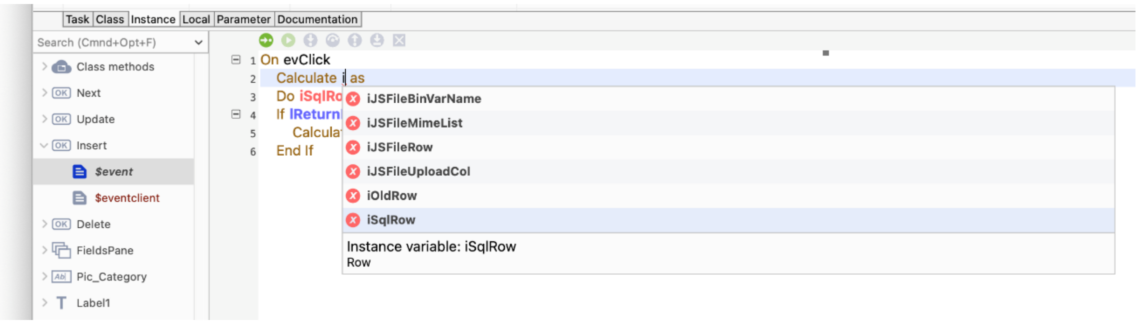
Description

This command assigns a new value to a data field or variable. The form of the command is "Calculate X as Y", where X is a valid data field or variable name and Y is either a valid data field or variable name, value, calculation, or notation. When **Calculate** is executed the state of the flag is unchanged, unless #F is recalculated by this command.

Figure 74:

- As soon as you select the *Calculate* command, the cursor is placed in the code line where a field or variable name is required. If you know the variable name, and in this case the column name, you can enter the text: **iSqlRow.Pic_ID** (this is the row variable and column name containing some data).
- Alternatively, you can use the **Code Assistant:** to do this, type the first letter of the variable, in this case "i", and select the **iSqlRow** variable name from the list that drops down

You can select the variable name with the pointer, or to keep your hands on the keyboard, use the **Up** or **Down Arrow** keys to move up and down in the list and press the **Return** key to confirm your choice, in this case select **iSqlRow.** This variable is contained in the remote form and was added automatically by the Remote Form Wizard – when the form is opened the variable will contain the current row of data.



```
On evClick
    Calculate i as
    Do iSqlRo
    If lReturn
        Calcula
    End If
```

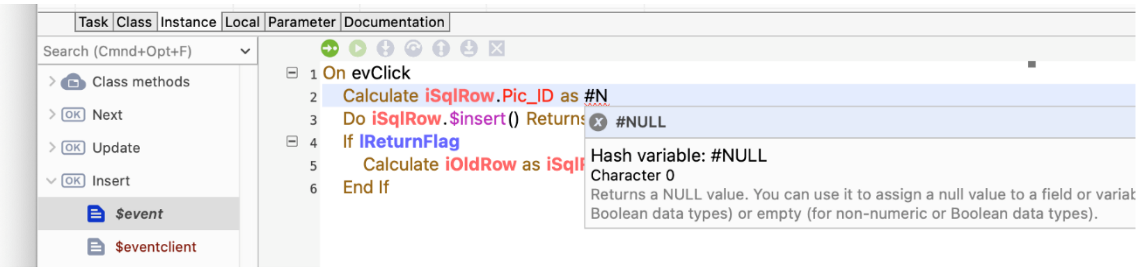
Instance variable: iSqlRow
Row

Figure 75:

- Having selected the variable name from the Code Assistant (or entered the name manually), type a dot (full stop/period), then enter the database column name, in this case, **Pic_ID** – the complete text should be: **iSqlRow.Pic_ID**
- Press the **Tab** key to move the cursor to the end of the line to enter the *Calculation***,** then enter **#NULL**: you can type "#" (Shift 3 or Option/Alt 3 on a mac keyboard) and the Code Assistant will display all the "hash variables", or in this case, type "#N" and use the **Down Arrow** and **Return** keys to select "#NULL" from the Code Assistant.

#NULL is a so-called "hash variable", a built-in Omnis variable (constant), and represents a NULL value. The line of code you added to the Insert button ensures that whatever value is entered by the end user into the field in the remote form, just before the row of data is inserted, the value of Pic_ID in the remote form is set to NULL and SQLite will insert a new value automatically (you may not have included the Pic_ID field on the form, in which case, this line of code is still needed to ensure a correct value for Pic_ID is entered into the database automatically). The complete code line is:

```
Calculate iSqlRow.Pic_ID as #NULL
```

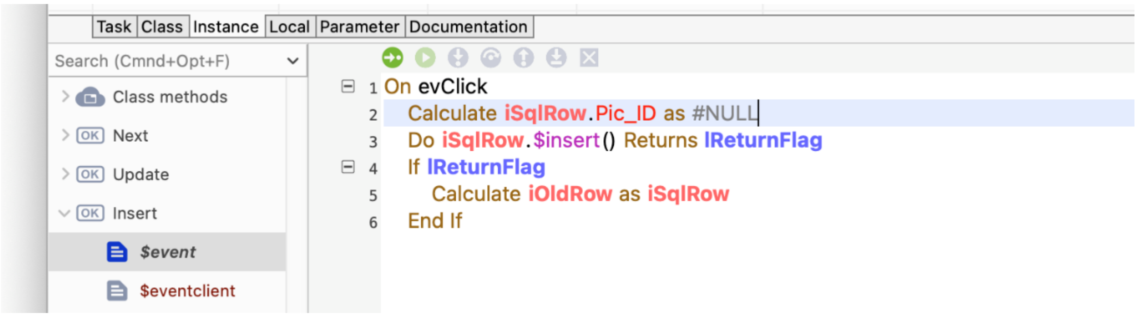


```
On evClick
  Calculate iSqlRow.Pic_ID as #N
  Do iSqlRow.$insert() Returns
  If lReturnFlag
    Calculate iOldRow as iSqlR
  End If
```

#NULL

Hash variable: #NULL
Character 0
Returns a NULL value. You can use it to assign a null value to a field or variab
Boolean data types) or empty (for non-numeric or Boolean data types).

Figure 76:



```
On evClick
  Calculate iSqlRow.Pic_ID as #NULL
  Do iSqlRow.$insert() Returns lReturnFlag
  If lReturnFlag
    Calculate iOldRow as iSqlRow
  End If
```

Figure 77:

Your method should now look like the following.

- When you have finished editing the method, close the Method Editor (by clicking on the close box), and then save your library using **Ctrl/Cmnd-S,** select **Save** from the **File** menu, or click on the **Save** button on the main Omnis toolbar.

Testing your Web Form

If you created your remote form from scratch it may look different from the JavaScript remote form created using the wizard (i.e. it only has a Next button), but testing a remote form is the same. Remember, if you have quit and restarted Omnis Studio, then you will need to reopen the PicSess database session, as described earlier in the 'Opening a Database Session' section.

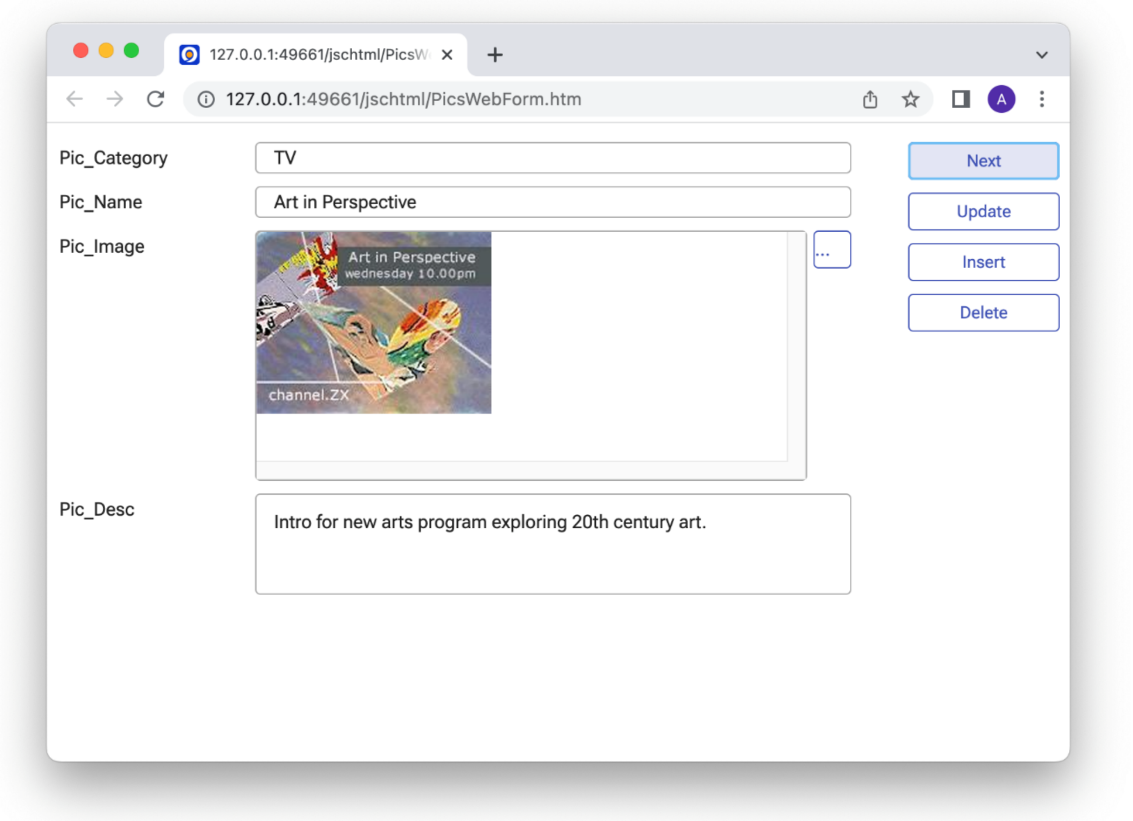
- If the **PicsWebform** form is not open in design mode, double-click on it in the **Studio Browser** to open it.
- To test a Remote form, you can click on the **Test** button in the Design bar in the Remote form editor
- Alternatively, to test a remote form you can press **Ctrl/Cmnd-T** at any time, or Right-click on the background of the form and select **Test Form.**

Your remote form should open in your *default web browser,* such as Chrome, Safari, Firefox, or Edge. However, you can test a Remote form in a different browser (other than your default web browser), by Right-clicking on the background of a Remote form and selecting the **Select Browser and Test Form** option, then choosing the browser.

- In your web browser, click on the **Next** button and each data record should be displayed.

If the data is not displayed this may be because the Pics database session is not open (maybe you closed Omnis Studio and returned to the tutorial later). If you need to open the database session, close the browser tab, return to Omnis Studio, click on **SQL Browser** in the Studio Browser, click on the *Open Session* option, then click on **PicSess** to open the database session, as described in the 'Opening a Database Session' section.

Having re-opened the PicSess session, return to your PicsWebform *in design mode,* click on the **Test** button (or press Ctrl-T) to open it in your web browser and try clicking the **Next** button again, or if the PicsWebform form is still open in your web browser you can return to the browser and **Refresh/Reload** the web page.



Pic_Category TV

Pic_Name Art in Perspective

Pic_Image

Pic_Desc Intro for new arts program exploring 20th century art.

Figure 78:

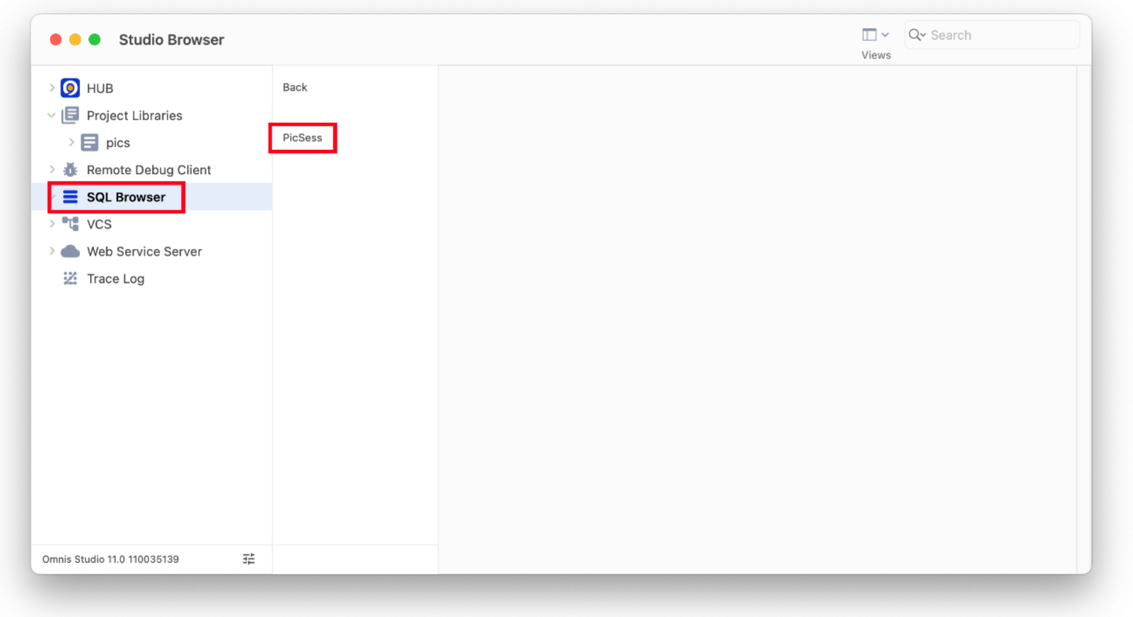


Figure 79:

Inserting some data

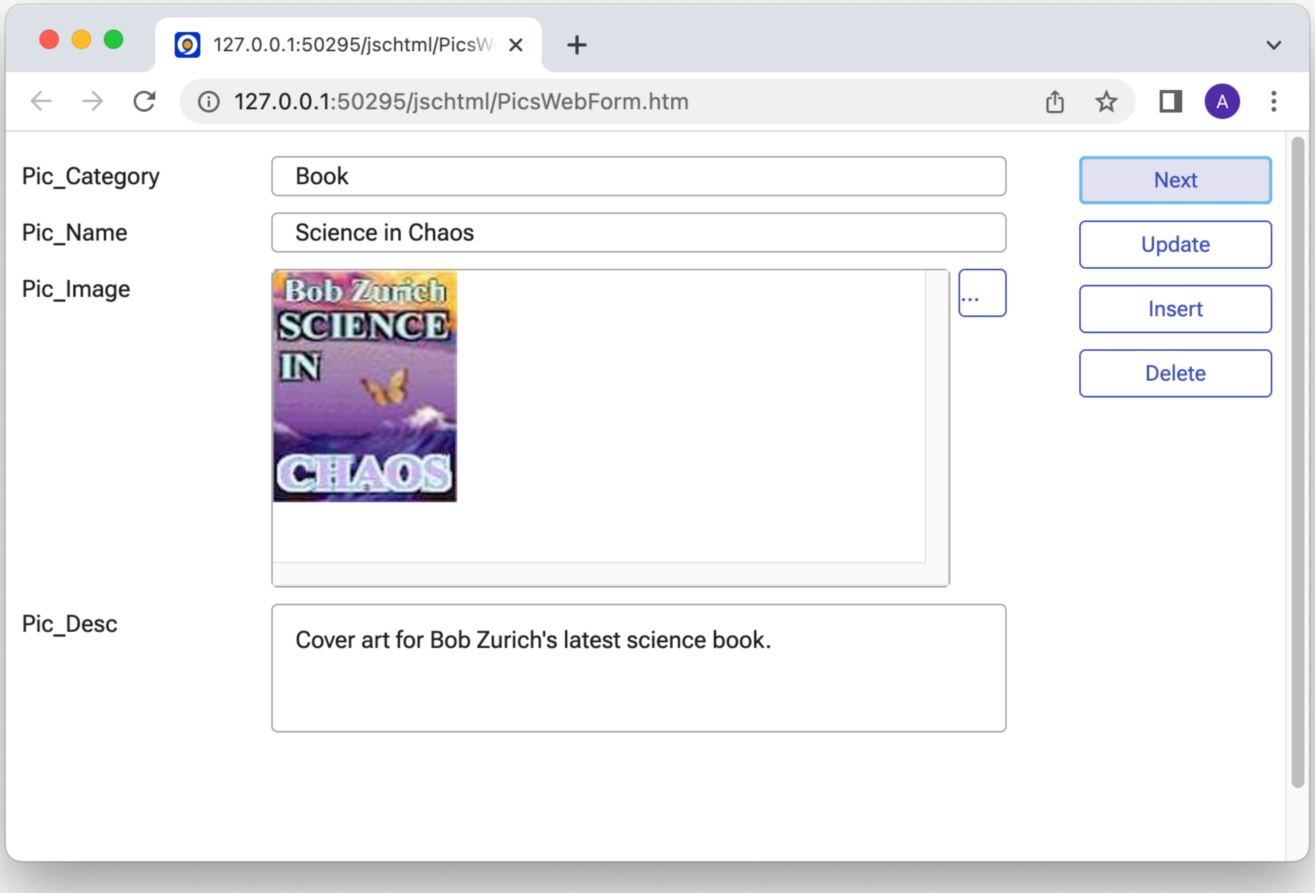
Assuming your PicsWebForm is open in your web browser (and that you have added the code to the Insert button, described in the previous section), then you can insert a new record into the database. To insert data you must ***first enter some data into the fields*** as follows:

Field	Data to add
Pic_Category	Book
Pic_Name	Science in Chaos
Pic_Image	Use the Upload button to insert the ‘11bookchaos.png’ from the tutorial folder you downloaded in the zip file
Pic_Desc	Cover art for Bob Zurich’s latest science book

Note that you did not include the Pic_ID and Pic_URL columns on the form; Pic_ID will be inserted automatically using the code you added to the Insert button.

- Next ***you must click on the Insert*** button to insert or “save” the record.

Click on the **Next** button to cycle through the data and you should see the record you just inserted.


127.0.0.1:50295/jschtml/PicsW
127.0.0.1:50295/jschtml/PicsWebForm.htm
Bob Zurich
SCIENCE
IN
CHAOS
...
Next
Update
Insert
Delete

Pic_Category: Book

Pic_Name: Science in Chaos

Pic_Image

Pic_Desc: Cover art for Bob Zurich's latest science book.

Figure 80:

Test HTML page

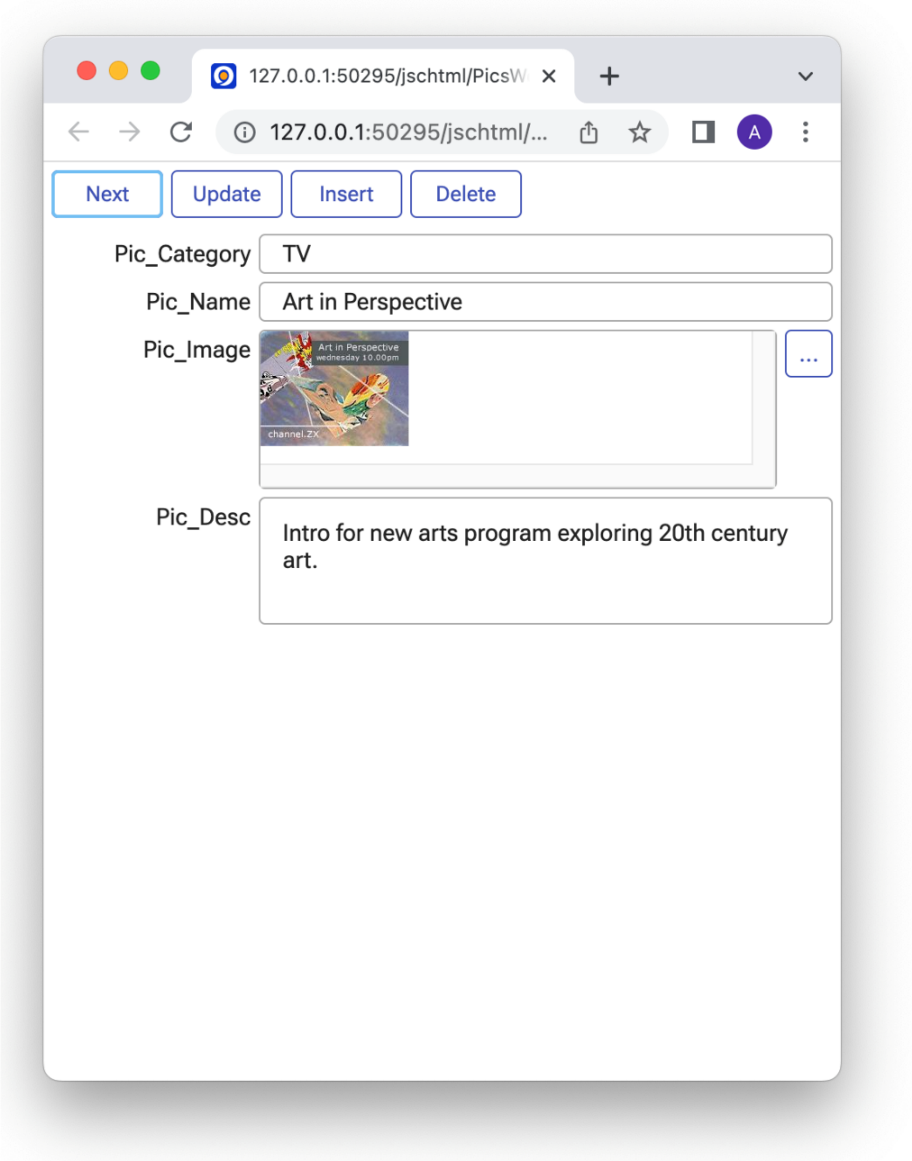
When you test a remote form, using the **Test** button (or **Test Form** option or **Ctrl/Cmnd-T**), Omnis creates an HTML page for you automatically, containing the JavaScript client; in this case the page is called ‘PicsWebform.htm’ and is placed in the ***html*** folder inside the Omnis Studio tree (the browser connects to the page using your ‘Localhost’ and a random test port setting). The test HTML page displays your remote form by connecting to your development version of Omnis Studio and the Pics library you have created.

You can view the source for the test HTML page in your browser to see the embedded JavaScript client object – look for the <div> with id=omnisobject1, and some of the other parameters that relate to the name of the library (data-omnislibrary=“pics”) and the remote form you created (data-omnisclass=“PicsWebForm”).

```
<div id="omnisobject1" style="position:absolute; top:0; left:0;
width:100%; height:100%;" data-webserverurl="_PS_"
data-omnisserverandport="" data-omnislibrary="pics"
data-omnisclass="PicsWebForm" data-themename="default"
data-appid="pics.PicsWebForm" data-dss="'js320x480Portrait',
'js768x1024Portrait'" data-param1="" data-param2=""
data-commstimeout="0"></div>
```

Viewing a different layout

Next you can resize your desktop web browser, that is, make it narrower, and the form will switch to display the layout for the 320 breakpoint. It will look something like this (if you used the Remote form wizard):



Pic_Category TV

Pic_Name Art in Perspective

Pic_Image

Pic_Desc Intro for new arts program exploring 20th century art.

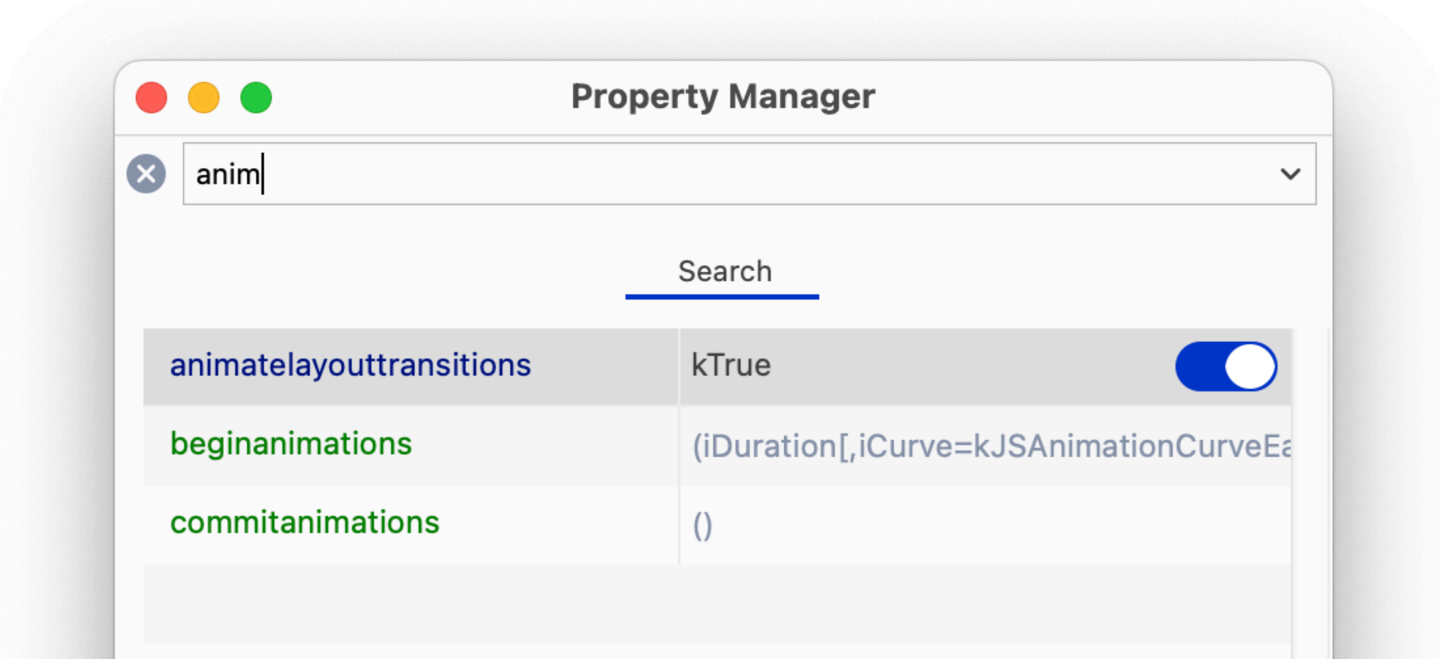
Figure 81:

You can animate the transition between the two layouts, by setting the **animatelayouttransitions** property.

- Close your web browser (or tab) and open the PicsWebform remote form in Omnis in design mode
- Click on the background of the form and open the **Property Manager** or bring it to the top by pressing F6/Cmnd-6
- ***Enable the Advanced option*** at the bottom left of the Property Manager to show all the properties and tabs

The **animatelayouttransitions** property is not shown under the basic view mode (when Advanced is disabled); when you switch to Advanced mode, all the properties and group tabs will appear in the Property Manager to categorize different types of properties. For a Remote form, you will see General, Appearance, Methods, and Action tabs.

- To locate the **animatelayouttransitions** property, you can click on the **Action** tab and select the property
- Alternatively, you can enter 'anim' or 'animate' into the property Search at the top of the Property Manager to find the **animatelayouttransitions** property.



animatelayouttransitions	kTrue
beginanimations	(iDuration[,iCurve=kJSAnimationCurveEa
commitanimations	()

Figure 82:

- Enable the **animatelayouttransitions** property (set it to kTrue).

If you want to locate other properties in the Property Manager, you need to clear the search by clicking the X icon next to the search, or deleting the search string.

- Open the remote form again in your web browser using the **Test** button (or Ctrl/Cmnd-T) and try changing the size of your desktop web browser; this time, the controls and buttons will glide smoothly between different layouts.

Viewing your form on a mobile device

Resizing your desktop web browser is a quick way to test the different layouts in a remote form, but you can test the form on a mobile device, or any other computer or tablet.

During development and testing you can load the form on any device that is within the same local network WLAN. To do this, you can enter a test URL into the web browser on your mobile or tablet, but replace the Localhost IP address (127.0.0.1) with the IP address of your development computer itself, that is, the computer that is running Omnis Studio and the Pics library.

First, you'll need to open the form in the desktop browser on your development computer to find out the test URL.

- If the Remote form is not already open, click on the **Test** button (or Ctrl/Cmnd-T) to open it.

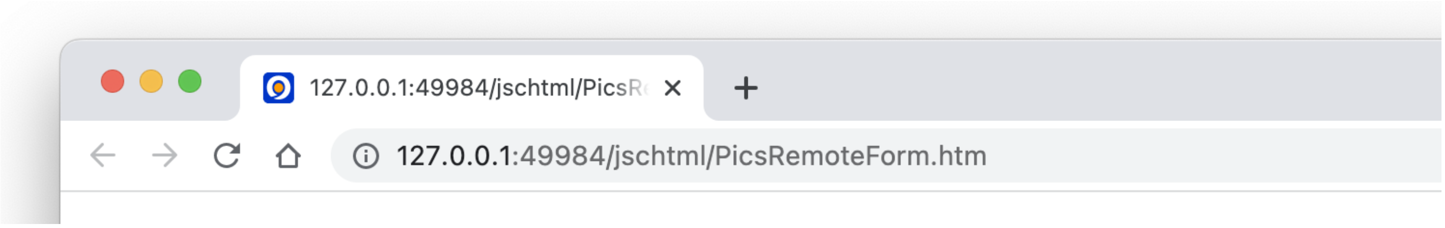

127.0.0.1:49984/jschtml/PicsR
127.0.0.1:49984/jschtml/PicsRemoteForm.htm

Figure 83:

Make a note of the URL in the web browser of your development computer, or just leave open the browser, and have ready your mobile phone or tablet, or any other computer within the current WLAN.

Next you need to find out the IP address of your development computer.

- On a **Windows** PC, you can use the 'ipconfig' command in the Command prompt to find your IP address, while on **macOS** you can use 'ifconfig' in a Terminal window.
- Open the web browser *on your mobile device or tablet* and enter the same URL displayed in the browser on your development computer, but *replace the local host IP* (127.0.0.1*) with the IP address* of your development computer. (You could try this in your desktop browser first and then send the link to your phone.)

The URL will be something like the following (the port number and IP address will be different, otherwise the URL should have the same format):

http://192.168.1.130:49984/jschtml/PicsWebform.htm

When you submit the test URL on your mobile device the PicsWebform remote form will open, but this time Omnis detects that the form is on a mobile device and displays the correct layout.

Changing the form theme

When you create a remote form a 'default' color theme is selected and applied to the controls and form background automatically, but you can change the theme in the remote quite easily. You need to return to your remote form in design.

- If the **PicsWebform** form is not open, double-click on it in the **Studio Browser** to open it; or if it is already open, bring it to the front.
- Press **Ctrl-J,** or select the **JavaScript Theme** option in the **View** menu.
- Select a different theme, and the colors used in the remote form will change; the **'vintage'** theme is selected in the following screen.

Note that the JS Theme is a global setting so the selected theme will be applied to all the remote forms in your library (or any other open libraries); you can write code in your remote form to set the JS Theme to be used on the client.

- Click on the **Test** button (or press Ctrl/Cmnd-T) to open the Remote form.
- Alternatively, you could *reload* or reopen the form on your mobile device to see how the form looks with a different theme; the following screen shows the mobile form with the Vintage theme.

Summary

This tutorial does not cover the process of deploying your application to the web, but hopefully it has given you an insight into how quick and easy it is to create and test remote forms in a web or mobile app using Omnis Studio.

For more information about deploying your web application, refer to the Deployment chapter.

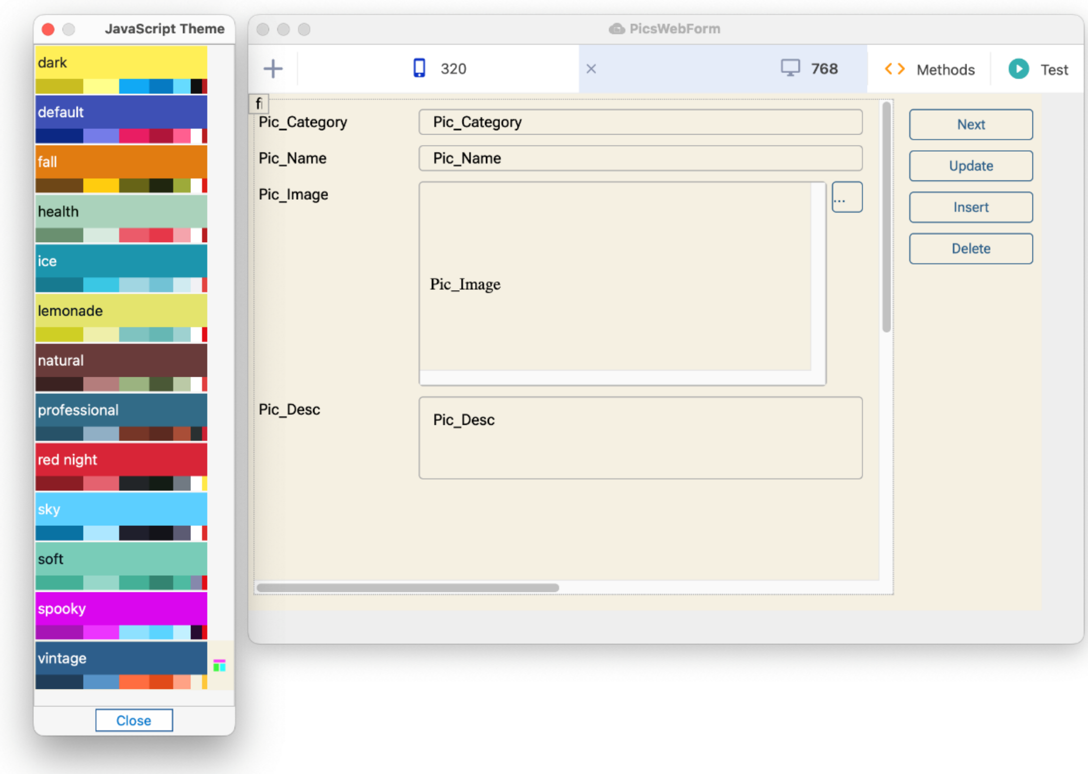


Figure 84:

Chapter 2—JavaScript Remote Forms

To create a web or mobile application in Omnis Studio, you need to create a **Remote Form** class using the **JavaScript Components** available in the Component Store in design mode. At any stage during the design process, you can test a Remote form by clicking on the **Test** button in the Design bar in the form, or by pressing **Ctrl/Cmnd-T.** Alternatively, you can Right-click on the background of a Remote form and select the **Test Form** option, to open the form in your default web browser, or you can select the **Select Browser and Test Form** option to select the browser in which to test the Remote form. The following screenshot shows the **jsCharts** Remote form in design mode in the JS Charts sample app:

When you **Test** (open) a Remote form in design mode it will open in a web browser on your development computer. The remote form you created will be opened in a simple HTML template file in your web browser using the *JavaScript Client* and rendered using JavaScript and CSS created for you automatically. This is the JS Charts remote form opened in a web browser (you can view this app under the **Samples** section in the **Hub** in the Studio Browser).

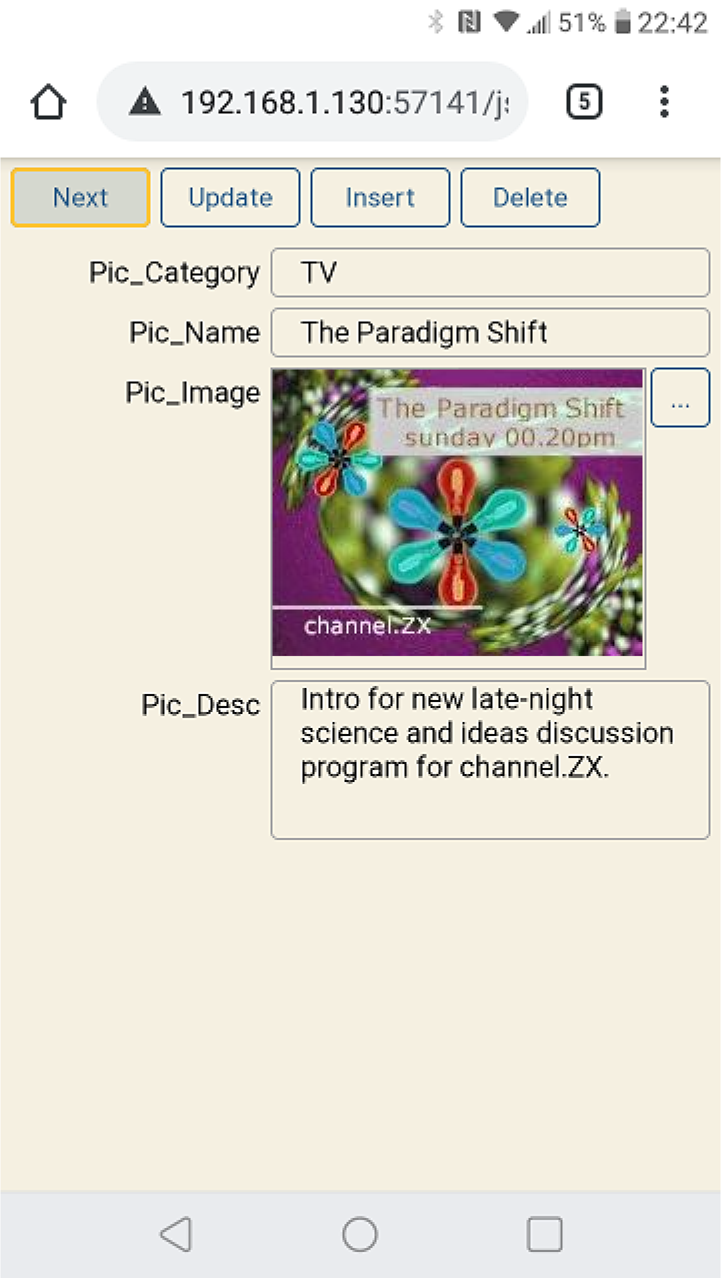
To test or open a remote form, your Omnis library also needs to have a **Remote Task** class which will handle the remote form instance(s) and any current connections to web or mobile clients. If your library does not contain a Remote task and you create a new Remote form (using a wizard, for example), Omnis will create a **Remote_Task** for you automatically; see Remote Tasks later in this chapter.

JavaScript Remote Forms and Remote Tasks are discussed in this chapter, while the individual *JavaScript Components* are described in the next chapter.

Creating JavaScript Remote Forms

You can create a new Remote form class in the Studio Browser using the **New Class>>Remote Form** option, or the **Class Wizard>>Remote Form** option. The first option creates a blank remote form ready for you to drop in JavaScript components from the Component Store, while the second option launches the Remote Form Wizard, which helps you build a complete form, step-by-step (based on an existing Schema or Query class, and a SQL session).

If you create a new library in the Studio Browser using the **Web and Mobile** option, the new library will contain a Remote form (and a Remote task) ready for you to add your own JavaScript Components.



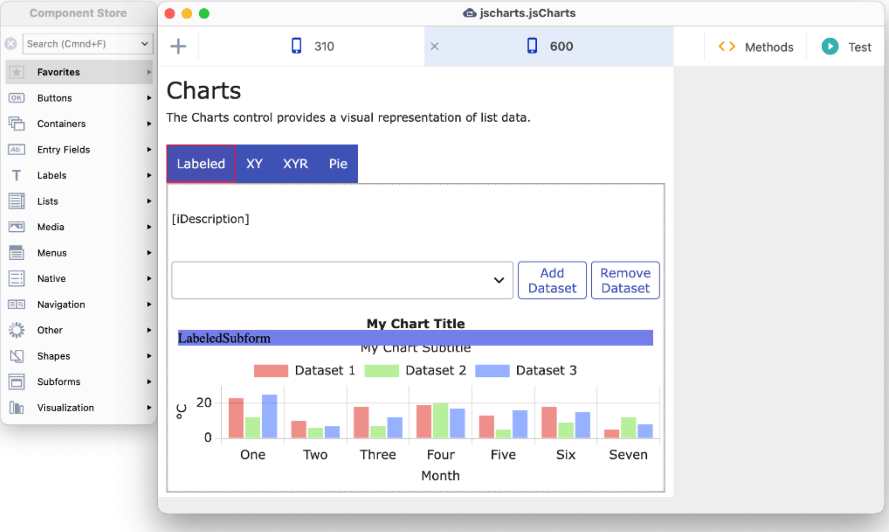
Pic_Category: TV

Pic_Name: The Paradigm Shift

Pic_Image

Pic_Desc: Intro for new late-night science and ideas discussion program for channel.ZX.

Figure 85:

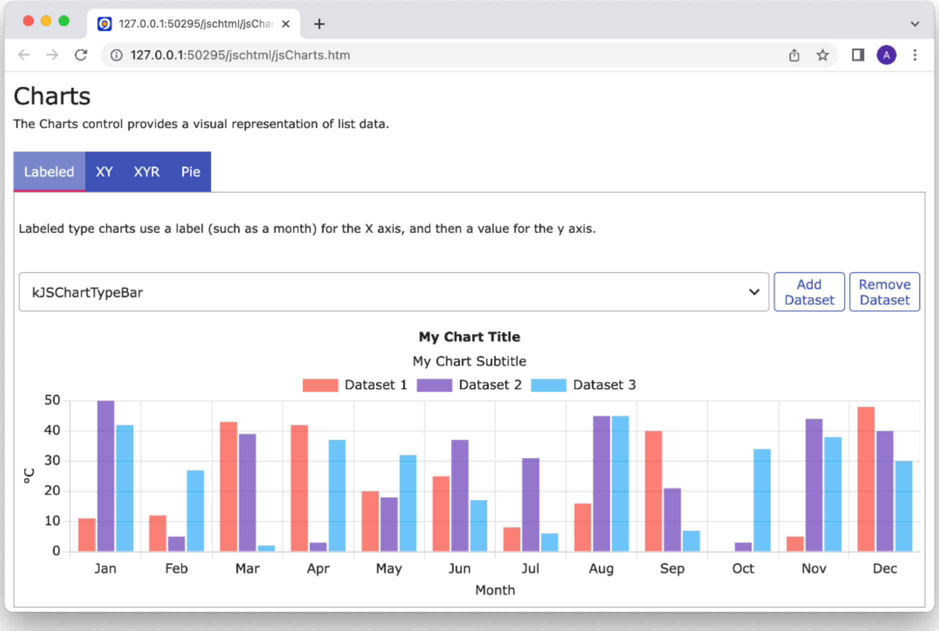


Charts

The Charts control provides a visual representation of list data.

[iDescription]

Figure 86:



Charts

The Charts control provides a visual representation of list data.

Labeled type charts use a label (such as a month) for the X axis, and then a value for the y axis.

kJSChartTypeBar

Figure 87:

Creating a Remote Form using a Wizard

You can very quickly create a Remote form to access and browse your database using the **Remote Form** Wizard. Before using the Remote form wizard, you need to create a schema class in your library, to match the data structure in your database, and you need open a session to your database. To create a Remote form using a wizard:

- Create and open a session to your database in the SQL Browser
- Create a Schema class based on your database table (drag a database table onto your library to create a schema class automatically)
- Select your library under the **Project Libraries** option in the Studio Browser, and click on the **Class Wizard** option
- Click on the **Remote Form…** option to launch the Remote Form wizard
- Select the **SQL JavaScript Form** wizard (the default option), and step through the wizard, selecting your SQL session and schema columns to include in the form

These steps are described in detail in the Tutorial, including the 'Creating a Web Form using a wizard' section, so you may like to work through those sections if you want to try the Remote form wizard.

Creating a Blank Remote Form

You can create a blank Remote form and add JavaScript components and code yourself.

- Select your library under the **Project Libraries** option in the Studio Browser, and click on the **New Class** option
- Click on the **RemoteForm** option
- Name the **New Remote Form** and press Return

Remote Form Name

Like any other class in Omnis, the name of a new Remote form can be anything you like, but the name would normally take account of its function within your application. The class name does not have to conform to any convention other than any conventions you may like to use in your application to identify different class types: so, for example, your remote forms could be prefixed with "js" (short for JavaScript), such as "jsCustomerForm".

You should note that the name of the Remote form class, plus the ".htm" extension, is used as the name of the HTML file which is created when you test your remote form in a web browser in design mode. Therefore, you should restrict any characters used in the name of your Remote form to only those normally allowed in a web context, or to be sure of removing all possible conflicts, you should only use ***alphanumeric characters*** and ***do not use spaces;*** you can use underscore to separate words if required. A remote form name cannot include the hash symbol (#) or other special symbols, since this may cause unexpected results in a web browser, or in the case of #, the remote form may not open in test mode at all. Omnis will warn you if you try to use any characters that are not allowed.

JavaScript Client

The JavaScript Client functionality is enabled by setting the **$client** property in a new remote form class. When you create a new remote form, either a blank one or using the Remote Form Wizard, Omnis will set the $client property to **kClientJavaScript** automatically, so the JavaScript Client is the default client for your web and mobile apps.

In previous versions, the kClientiOS and kClientPlugin values of $client were available, but these options have been removed from the Property Manager; in effect, the kClientJavaScript option is now the default or only possible value of $client.

JavaScript Components

When you edit a remote form in design mode, the **Component Store** will be displayed, showing different JavaScript components in different groups. The following screenshot shows the **Buttons** group containing a number of related components, including the Button, Check box, and Switch controls.

See the *JavaScript Components* chapter for more information and example code for each JavaScript component.

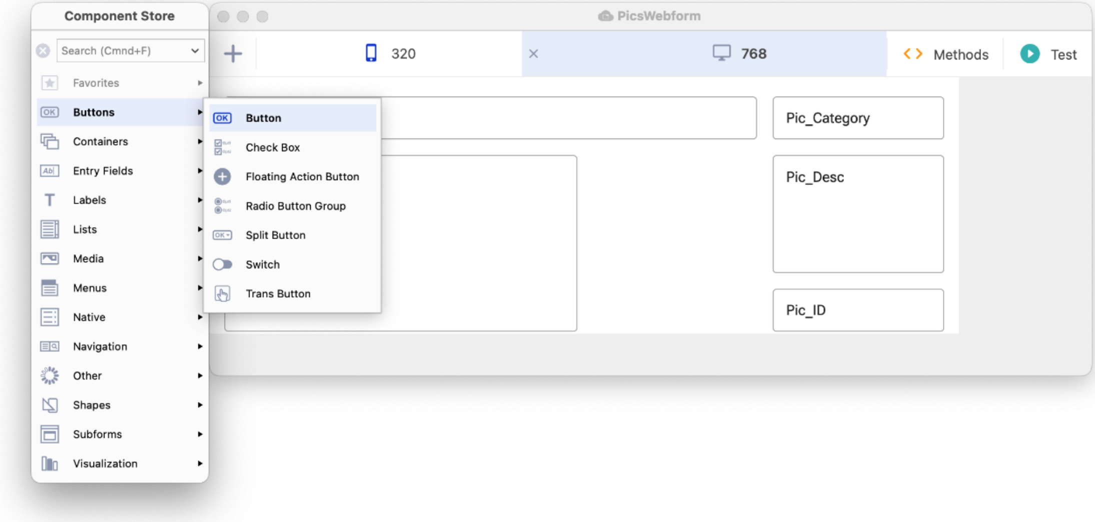


Figure 88:

Remote Form Design

When you create or modify a JavaScript Remote form class, the form design window is displayed in a ***Web Preview*** using the Chromium web browser built into Omnis (using the Chromium Embedded Framework or CEF), so you can see how your form will look at runtime in the end user's web browser. Specifically, JavaScript controls (and JSON-defined controls) will look virtually the same in design mode as they will do at runtime in a web browser, including the visual effect of any CSS styles you have applied to the controls (using $cssclassname). In addition, your remote form and its controls will be displayed using the current JavaScript theme, which you can change by pressing Ctrl-J and selecting a different theme: see JS Themes.

To render the Web Preview, an HTML file is generated using the jsctempl.htm template file in the 'html' folder in the Omnis tree, or the file named in the $htmltemplate property in the design task, and placed in the 'html/design' folder; note the files in this folder are only used for rendering the Web Preview in design mode and are not required when you deploy your application.

Note to pre-Studio 10.2 users: Using old design mode

You should note that there are a few differences between the Web Preview mode for remote forms and the form design mode in previous versions, as follows:

- There is no *design grid* available in the Web preview mode, so $showgrid is not present ($showgrid is not available if you switch to the old design mode).
- *Rulers* are not supported in the Web preview mode, so the remote form context menu does not have an option to show Rulers; you can instead use Position Assistance to lineup and size the controls on your remote form.
- *Design DPI scaling* does not apply in the Web preview mode.
- The JS client uses box-sizing border-box, so the appearance of control borders may be different.
- It is possible an exception will occur in the JS client running in the new Web preview mode: this does not have any effect on the validity of the remote form class. If this occurs, a message will be displayed for 5 seconds, and the error will also be logged to the trace log. In this case, you should close and re-open the remote form editor after an exception.

Remote Tasks

To test or run your remote form in a web browser, ***your library must contain* a *Remote Task*** and the $designtaskname property of your Remote form must be set to the name of a Remote task in the current library. When you create a new library or Remote form, Omnis will in most circumstances create a Remote task for you; the following points summarize how this is handled:

- If you create a new library in the Studio Browser using the **Web and Mobile** option, the library will contain a NewRemoteForm and a Remote_Task and the $designtaskname property of the form will be set automatically.

- If you create a new Remote form in an empty library (i.e. without an existing Remote task), and try to Test the form, a new Remote_Task will be created automatically and the $designtaskname property will be set to that task.
- The Remote form wizard (SQL JavaScript Form) creates a new Remote_Task if a task class does not already exist in your library and sets the $designtaskname property of the form automatically.
- If you try to open a remote form without a remote task in your library, or without $designtaskname being set to a remote task name, Omnis displays an error message and the form will not open.

If your library contains multiple remote tasks, and you create a new Remote Form from the Studio Browser (using the New Class option, not the wizard), the $designtaskname property *will not be set,* so you will have to assign the design task name manually before you can test the remote form.

Creating Remote Task Classes using Wizards

For the purposes of prototyping and testing your web or mobile application, you can use the Remote tasks created for you automatically. You will need to edit the Remote task when you are ready to deploy your application, or if you want finer control over the processes in your web application. However, if you want to create a Remote task, you can use one of the templates or wizards provided in the Studio Browser.

To create an empty remote task

- Click on the **New Class** option in the Studio Browser and then the **RemoteTask** option
- Name the new Remote task and press Return

A suitable alternative would be the Plain remote task, described below.

To create a remote task using a class wizard

- Click on **Class Wizard** option in the Studio Browser and then the **Remote Task** option
- Select the wizard you want, name the new class and click on **Create**

The following wizards/templates are available:

- **Plain Remote Task**
  creates an empty remote task containing an $event method with code for evBusy and evIdle events; this is suitable for running JavaScript Remote forms.
- **Monitor Remote Task**
  creates a task and remote form/window to monitor remote connections from within your application when deployed on the Omnis App Server.
- **HTML Report Task**
  creates a task to generate HTML reports on the fly
- **Submit_Task**
  creates a task and standard HTML file containing a submit form which interacts directly with Omnis; note the HTML file created using this wizard does not use the JavaScript Client.

$enablesenddata Property

Existing users should note that the remote task property $enablesenddata should be set to kFalse for all remote tasks controlling JavaScript remote forms since the $senddata() method is not implemented for JavaScript remote forms. Therefore, this property is not necessary in new web or mobile applications using the JS client and may not be shown in your version of Omnis Studio.

Plain Remote Task Wizard

The Plain Task wizard creates a basic template remote task that is suitable for linking to most simple remote forms. The Plain remote task also has an $event() method containing a template event handling method that detects evBusy and evIdle events in the task. You can add your own code to handle these events.

The Plain Remote Task has a $construct() method containing a parameter variable called *pParams* of type *Row Variable*. This row variable receives all the parameters of the JavaScript Client, such as the remote form name, the client width and height, and the user agent string: see below.

When you create a task using the Plain Task wizard you can specify the **Inherit from Monitor task** option. This option adds a set of "monitor" classes to your library which allows you to record client connections associated with the new plain task you are adding to your library. If you check the Monitor option, the wizard prompts you for details about the new monitor task. If your library does not contain a monitor task, you need to specify the **Create New Monitor Task** option. If, however, your library contains a monitor task, you can specify the **Use Existing Monitor Task** option to add the new plain task you are currently adding to your library to the existing monitor.

Monitor Remote Task Wizard

The Monitor wizard (or checking the 'Inherit from Monitor task' option in the Plain Task wizard) creates a number of "monitor" classes, including a new task and monitor Remote form and /or Window class, that allow you to record remote connections between web or mobile clients and your application hosted on the Omnis App Server.

The wizard prompts you to enter the name of the new Monitor *remote task* and *remote form* for displaying the connection results and activity: alternatively, the wizard can create a desktop window class, or both the remote form and window classes. In addition, an extra pane allows you to identify the remote task in your library that needs to have the Monitor set for its superclass (this had to be set manually in versions prior to Studio 10.x).

The **Amend Startup Task** option lets you add code to the Startup_Task in the current library to open the Monitor form/window at startup; this is checked by default.

The Monitor form/window has three panes. The **Connections** pane shows the connections grouped by remote form name. The **History** and **Server Usage** panes let you monitor the traffic flow on your Omnis App Server and provide some general information about server usage. You can print the server usage using the **Print Report** button.

Server Management Library

The Server Management Library contains the same monitor classes that you can use to monitor an instance of the Omnis App Server; the servermgmt.lbs library is located in the /webserver folder under the main Omnis folder.

On Windows, the Server Management Library can display a tray icon; to enable the icon, you need to add the "showTrayIcon" item to the "servermgmt" section of the config.json file and set it to true. If omitted, or set to false, the servermgmt.lbs does not show a tray icon.

Remote Task Instances

When the JavaScript Client first connects to the Omnis App Server, Omnis creates an instance of the Remote Task Class associated with the Remote form class to which the client is connecting (and specified in the $designtaskname property of the remote form). Once the remote task class has been instantiated, next the Remote form instance is created. The $construct method of first the Remote task and then the Remote form are run, so these methods can include any code you want to run prior to opening the form (in the task construct method) or when the form is opened.

$order property

The $order property is an integer that uniquely identifies the remote task instance within the lifetime of the Omnis Server (since it was started). The value will not be re-used for a different remote task until the Omnis Server is restarted. Also, values are unlikely to be incremental.

Construct Row Variable

When a form is opened and the Remote task and Remote form instances are created, Omnis passes a parameter variable of type Row to the $construct() method of the Remote task and then the Remote form (in that order); this is called the **Construct Row** parameter variable. The row variable contains a column for each parameter of the *JavaScript Client Object* instantiated on the client: therefore, it will include columns for OmnisLibrary name and OmnisClass name (as defined in your HTML file), as well as extra columns containing additional information about the client object.

There is an example app called **Construct row** in the **Samples** section in the **Hub** in the Studio Browser showing how you can return information from the construct row; the same app is available in the JavaScript Component Gallery.

The construct row variable will contain the following columns and typical values:

Column	Description
OmnisLibrary	<OmnisLibrayName> minus the .lbs extension
OmnisClass	<RemoteFormName> e.g. jsRemoteForm
appid	<OmnisLibrayName>.<RemoteFormName>
param1, 2, .. 9	Up to 9 pre-defined custom parameters called param1, param2, etc, which receive the values in the parameters added to the JavaScript Client object in your HTML page; you can add custom parameters prefixed with “data-“ to send further values to the remote task or form $construct method, e.g. data-param1=”123” data-param2=“abc”, etc.
OmnisPlatform	JSU, the JavaScript Client
JSscreenWidth	The screen width of the client, e.g. 2048 for desktop
JSscreenHeight	The screen height of the client, e.g. 1152 for desktop
JSscreenSize	The initial setting of $screensize (only applies to kLayoutTypeScreen based forms in Studio 8.0 or earlier, so not relevant for responsive remote forms)
JSDeviceInfo	Device screen size (Width x Height) and density, e.g. 1920x1080 (devicePixelRatio:1) for desktop with standard monitor, 360x640 (devicePixelRatio:2) for mobile phone with HD display
clientPlatform	The platform on which the client is running, one of the following strings: ‘Windows’, ‘macOS’, ‘Linux’, ‘iOS’, ‘Android’ or ‘Unknown’
userAgent	The navigator.userAgent of the client, which usually contains the browser type and version (e.g. Mozilla/5.0)
appName	The navigator.appName of the client, i.e. the browser application name, e.g. “Netscape” (or “Microsoft Internet Explorer” for older clients)
Flags	Currently indicates if the client supports animation: 1 means the browser does support animation, zero means that it does not

Column	Description
JStimezoneOffset	offset from UTC in seconds, e.g. “60” for clients on UTC+1, “120” for clients on UTC+2, etc.
ClientLocale	the Locale language setting of the client, e.g. “en_GB” for clients in the UK, or “en_US” for America
theme	The current JS theme as set in the $javascripttheme Omnis preference (e.g. ‘default’)
URLparams	One or more parameters added to the URL for the web page containing your remote form; see below. For example, ?x=y&a=b appended to the URL are returned as a JSON object string {“x”:“y”,“a”:“b”}
window_	specified as data-window, a comma-separated list of members of the JavaScript ‘window’ object; see below
localpref_	specified as data-localstorage, a comma-separated list of preference names saved to localStorage; see below

The appName and userAgent columns return properties of the client browser and therefore allow you to determine which browser and version the client is using, such as whether it is a desktop or mobile browser.

Using the Construct Row Variable in your Code

If you want to use the values in this parameter variable, you can create a parameter variable of type Row in the $construct() method of your remote task or remote form which will receive the parameter variable when the task/form is constructed. To examine the values in the variable, you can set a breakpoint in the $construct() method of your remote task or remote form, open the form (using Ctrl-T), and Omnis will switch to the debugger allowing you to right-click on the variable to examine its value.

The following example uses the screen size of the client device to set the size and position of various controls in the initial remote form for the app. The $construct() method of the remote form receives the pRow parameter row variable containing the screen size of the client device, and calls another method to setup the columns for a data grid control on the main remote form:

```
# $construct method containing a Parameter var called pRow of type Row; the form also
# contains an instance var iScreensize (Char)
Calculate iScreensize as pRow.JSScreensize
Do method setupSizes
Etc.
# code for setupSizes method
Switch iScreensize
Case kSSZjs320x480Portrait
  Do $cinst.$objs.pagePane.$objs.orderGrid.$::columnwidths.$assign("150,50,50,70")
Case kSSZjs320x480Landscape
  Do $cinst.$objs.pagePane.$objs.orderGrid.$::columnwidths.$assign("70,40,40,70")
Case kSSZjs768x1024Portrait
  Do $cinst.$objs.pagePane.$objs.orderGrid.$::columnwidths.$assign("300,75,75,175")
Case kSSZjs768x1024Landscape
  Do $cinst.$objs.pagePane.$objs.orderGrid.$::columnwidths.$assign("250,75,75,175")
Default
  Do $cinst.$objs.pagePane.$objs.orderGrid.$::columnwidths.$assign("100,50,60,75")
End Switch
```

Passing Additional Parameters via a URL

You can pass additional parameters to a remote task (or remote form) from the JavaScript Client by adding the parameters to the URL for the web page containing your remote form. This is in addition to the parameters that can be sent to the remote form or task in the construct row variable, and any that may be quoted in the HTML page containing your remote form using the data-param1, data-param2,.. tags.

The additional parameters can be appended to the URL pointing to the remote form in the following format:

```
http://127.0.0.1:5988/jschtml/rfSetCurField.htm?x=y&a=b
```

The JavaScript client adds the parameters as an optional column called URLparams in the row variable passed to the $construct() method of the remote form and remote task. The data in URLparams is encoded as a JSON object string, e.g. if the URL params are x=y&a=b, as above, the JSON object string has the value {“x”:“y”,“a”:“b”}. You can use the new OJSON static function to convert this to a row:

```
Do OJSON.$jsontolistorrow(pRow.URLparams) Returns lRow
```

where lRow is a row variable. For the JSON above, the value of lRow.x will be ‘y’ and lRow.a will be ‘b’. Note: the client also decodes any special encoded URI characters before generating the JSON, e.g. %3D will become =.

Sending Data to the Form construct

You can send data or content to the $construct method of a remote form by specifying some extra attributes in the Omnis JavaScript object, contained in a <div> called “omnisobject1”, as follows:

- **data-localstorage**
  A comma-separated list of preference names saved to localStorage (e.g. using the ‘savepreference’ $clientcommand), whose values should be sent to the $construct row in the form. They can be named “localpref_<prefName>”
- **data-window**
  A comma-separated list of members of the JavaScript ‘window’ object, whose values to send to the $construct row in the form. You can use dot notation to access nested children. The columns returned to Omnis will be named “window_<memberName_childName_...>”. Column names have a max length of 255 characters

For example, the following parameters added to the omnisobject (shown in bold) will send the pixel ratio of the current device, plus the myPref1 and myOtherPref parameters from local storage to the $construct of the remote form:

```
<div id="omnisobject1" style="position:absolute;top:0;left:0"
data-webserverurl="" data-omnisserverandport="" data-omnislibrary=""
data-omnisclass="" data-dss="" data-param1="" data-param2=""
data-commstimeout="0" data-window="document.URL,devicePixelRatio"
data-localstorage="myPref1,myOtherPref"></div>
```

Class Cache Logging

You can log and control the caching of classes in the JavaScript Client. For most applications, you should not need to use the cache logging and control, since the default behavior of caching all class data to localStorage provides the best performance, and is adequate for most remote forms and data.

The options are only provided if you find your application reaches the limits of localStorage (e.g. with a very large application) and you need to examine and control the contents of the cache.

If you have reached the localStorage limit, and need to manually clear the cache, you can do so by running the following JavaScript code in your browser:

```
localStorage.clear();
```

To enable the cache logger, the omnisobject <div> can have two optional attributes:

- **“data-logcaching”**
  If present, data will be collected on the caching of class data, etc in localStorage.
  This can be accessed by querying the JavaScript object jOmnis.omnisInsts[0].cacheLogger. It has methods getCacheLog() and printLocalStorage() to provide useful information in the browser console. If given the value “verbose”, it will print caching messages to the console as they occur.

- **“data-onlycacheclasses”**
  If present, cache only the class data for the specified classes in localStorage.
  A comma-separated list of Remote Form classes whose data should be cached.
  In the format “<library name>.<form name>”. E.g: “myLib.jsForm1,myLib.jsForm2”
  #STYLES is handled separately, per-library. To enable caching of styles, add an entry “<library name>.#STYLES”

These parameters will need to be added to or enabled in the HTML page containing the initial remote form for your web or mobile application (they could also be added to enabled in the jsctempl.htm file, although the cache logging does not need to be enabled for most applications).

Changing Forms

The remote task instance has a method, **$changeform(),** which enables you to replace the form currently in use on the client, with another form in the same library. $changeform() has a single argument, which is the new remote form name. When it executes, *the current form instance destructs,* and the client constructs an instance of the new form to display in the user’s browser. You can use task variables in the remote task instance to pass information between the destructed remote form instance, and the new remote form instance.

There are some restrictions to note:

- $changeform() cannot be used in the $construct() or $destruct() method of a remote form instance or remote task instance. If used, Omnis generates a runtime error.
- Multiple calls to $openform() (described later) or $changeform() during the processing of a single event will result in only the last call to $openform() or $changeform() having any effect.

One scenario for using $changeform() is where the end user is required to log onto your web application, whereby the initial “logon” form prompts the user for a name and password, and the application changes to another form when the user has successfully submitted a valid name and password.

Multiple Forms

You can open more than one form within a single client connection, that is, within a single remote task instance. At any one time, only one of these multiple instances is visible, and the forms must be from the same library.

There are two methods of a remote task instance which you can use to manage multiple forms: **$openform()** and **$closeform()**. Like $changeform(), both these methods take a single argument, the remote form name.

If the form passed to $openform() already has a remote form instance open in the context of the remote task instance, it becomes the visible form for the remote task. Otherwise, Omnis constructs a new instance of the remote form in the remote task, and makes the new remote form instance the visible form.

The $closeform() method closes (destructs) the remote form instance for the named form, without closing the task instance or any other forms that may be open within the task. It is possible to close the last remaining remote form instance, but this is not recommended, since the end user will be presented with a blank screen. If the referenced form is not visible, the client observes no effect; otherwise, the most recently visible open remote form instance becomes visible.

There are some further restrictions to note:

- $closeform() and $openform() cannot be used in the $construct() or $destruct() method of a remote form instance or remote task instance. If used, Omnis generates a runtime error.
- Multiple calls to $openform() or $changeform() during the processing of a single event will result in only the last call to $openform() or $changeform() having any effect.
- Calling $showurl() or $showmessage() in the $destruct() method of a remote form has no effect.

- All forms must be in the same library.

You can use task variables to handle communication between multiple remote form instances in a remote task instance.

To facilitate communication between different remote form instances, remote forms can also receive the event evFormToTop. In design mode, you can enable this event for a form, using the $events property of the form. The event generates a call to the $event() method of the remote form. evFormToTop occurs when an existing remote form is about to become visible on the client as a result of a call or calls to $openform() or $closeform().

Client Access Properties

Remote tasks have a number of properties for managing the connections between the Omnis App Server and the web or mobile clients connected to your application. These properties will be populated only when there are live remote task and remote form instances created by a client connection.

- **$connectbytessent**
  specifies the number of bytes which have been sent to the client during the connection. This property is set after $construct() has been executed.
- **$requests**
  specifies the number of events executed on the server. Excludes connect and disconnect messages. Updated prior to evBusy message.
- **$reqtotbytesreceived**
  the total number of bytes received from the client for all requests. To calculate an average per request, you can divide this value by $requests. Updated prior to evBusy message.
- **$reqtotbytessent**
  the total number of bytes sent to the client for all requests. To calculate an average per request, you can divide this value by $requests. Updated prior to evIdle message.
- **$reqmaxbytesreceived**
  The largest block in bytes received from the client for all requests. Updated prior to evBusy message.
- **$reqmaxbytessent**
  The largest block in bytes sent to the client for all requests. Updates prior to evIdle message.
- **$reqcurbytesreceived**
  The number of bytes received from the client for the current request. Updated prior to evBusy message for the current request.
- **$reqcurbytessent**
  The number of bytes sent to the client for the current request. Updated prior to evIdle message.

Timeouts

You can control how long someone is connected to the Omnis App Server and how long a single client connection can remain idle, using the following properties.

- **$maxtime**
  the maximum time in minutes that a client is allowed to stay connected; the default value is 0 which means the client can stay connected indefinitely.
- **$timeout**
  the maximum time in minutes that a client is allowed to stay idle; the default value is 0 which means the client is allowed to stay idle indefinitely.

Client Connections

Remote tasks have some properties that tell you about the current client connection.

- **$clientaddress**
  the TCP/IP address of the current client. Note that this may not be the exact TCP/IP address of the client machine, due to the possible presence of proxy servers and firewalls between the client machine and the web server.
- **$connectionid**
  the id of the current client connection; ids are allocated dynamically by the Omnis Server and numbers are not reused unless the server is restarted.
- **$connectiontime**
  the time and date the client connected to the Omnis Server, i.e. the time the current task instance was instantiated.
- **$lastresponse**
  the time and date the client last accessed the remote task instance on the Omnis Server.

Managing Timeouts in Remote Tasks

Remote Tasks can be 'suspended' to allow greater control over how client connections are managed. A task may (optionally) be suspended if the web page is sent into the browser's persistent cache, or if the page becomes hidden (e.g. the user switches tabs).

When a task is suspended, it can automatically transition to a shorter timeout. An event is also fired on the task, so you might also want to take this opportunity, for example, to close your database or push connections.

A benefit of this is that it much improves the chance that Omnis will receive some kind of notification that mobile apps have gone away or have been killed by the user/OS, and will not leave the Remote Task open indefinitely.

Properties

Remote Tasks have the following properties:

- **$suspendconditions**
  A set of zero or more kSuspendCondition... values to indicate under which circumstances the client should tell the server to suspend the task.
- **$suspendedtimeout**
  The time (in minutes) the task will survive for while suspended. Zero means never suspend the task (the default) and -1 means suspend, but use the value of $timeout

The conditions under which the client may suspend are:

- **kSuspendConditionCache**
  The browser has stored the full page, including its state, in its back/forward cache. Support for this varies by browser (Chrome does not seem to support it), but it generally occurs when the user navigates away from the page using the browser's back/forward navigation buttons.
  Note: Fields with an $autocomplete property set to "off" may be cleared when the client is sent to the cache.
- **kSuspendConditionInactive**
  The page is no longer visible. E.g. the user has changed tab, minimized the browser or switched desktop.

If the Task times out while the client is suspended, you will receive a "You have been disconnected..." message on resuming. You can override this, as usual, by implementing a client-executed "$ondisconnected" method on your form, which returns true.

Important Note: The HTML templates contain support for this new mechanism (introduced in Studio 10.1), therefore you need to update any existing .htm files on your web servers to match, otherwise you will get errors or leak Remote Tasks.

Remote Tasks Events

Remote Tasks have the following events to handle task suspension:

- **evSuspended and evResumed**
  which will be called when the client is suspended or resumed, respectively. Both events receive a pSuspendCondition parameter with a **kSuspendCondition** value to indicate whether the client was suspended to the browser's cache or the page was hidden.

Remote forms Events

When the client is sent to/resumed from the cache or becomes hidden/visible again, an attempt will be made to call a *client-executed form* method named **"$suspended"** or **"$resumed"** on your main form.

This happens regardless of whether the Remote Task is actually suspended, so can be made use of in serverless-client apps, or if you just want to react to the page becoming visible again without using the suspend functionality.

These methods receive the following parameters:

- **pSuspendCondition**
  A kSuspendCondition… value indicating whether this event is occurring due to the page's visibility changing, or sent to the cache.
- **pTaskSuspended**
  A boolean indicating whether the Remote Task was/will actually be suspended. (It may not, depending on the Remote Task's $suspend… properties)

Secure Sockets

You can use secure sockets (HTTPS) if you have installed an SSL certificate on your web server. The JavaScript Client will use a secure connection to connect the client to the web server if you prefix the URL or IP_address in the data-webserverurl parameter with "https://". In addition, remote tasks have the $issecure property that lets you turn secure mode on and off dynamically, by assigning to the property for the current task at runtime.

Remote Task Events

For remote tasks, the **evBusy** and **evIdle** events are sent to the $event() method during the lifetime of a connection: evBusy is sent when Omnis receives a request from a client, evIdle is sent when Omnis is about to return the result of a request, i.e. the task instance is about to become idle. The following example, shows the code for the $event() method in the Monitor task created using the Monitor task wizard:

```
On evBusy
  If iMonitorOpen
    Do iMonitorRef.$setstatus($cinst,kTrue) Returns lServerBusyFlag
      If lServerBusyFlag
        Quit event handler (Discard event)
        ; server cannot handle request
    End If
  End If
On evIdle
  If iMonitorOpen
    Do iMonitorRef.$setstatus($cinst,kFalse)
  End If
On evRejected
  Do $cinst.$showmessage(pErrorText)
```

In addition, tasks report the **evRejected** event which is generated when Omnis rejects a connection by a client. Usually this occurs if there are too many users trying to connect to Omnis, or $maxusers of the remote task has been exceeded. The parameter pErrorText is "Too many users connecting to server" for the first case, and "Too many users connecting to task [taskname]" for the second.

Push Connections

Under normal operation, the Omnis Server cannot initiate communications with the client – all communications must originate as a request from the client. However, you can "push" data to the client using *Push Connections* by creating a web socket connection to the client. An example use-case could be that you could start off a long query using a SQL worker on the server, and then push the response to the client when the results are ready, updating any instance variables in the remote form.

Support for push connections has been implemented via a Long Polling mechanism called Pollymer, a general-purpose AJAX/long-polling library, since it provides a simple HTTP based solution that is supported in all browsers.

There is an example app called **HTTP Push** in the **Samples** section in the **Hub** in the Studio Browser showing how you can use push connections, and there is a Tech note describing how to use the example: 'REST Web Services HTTP JavaScript Push Example' TNWS0005.

Creating a push connection

Each JavaScript client remote task in Omnis can have a single "push connection", established using the client command **openpush.** The syntax is:

```
$cinst.$clientcommand("openpush",row())
```

The openpush client command can be executed in either a server or client executed method, but you are advised to use it in a server method to gain greater control over when the results are pushed. That way, you know exactly when you are using a push, or whether or not you want to push data. There is a matching client command, **closepush,** which you can use to close the push connection.

Utilizing REST

The push connection uses Omnis RESTful support to carry its requests, therefore, if you are using a Web server to pass JavaScript client requests to the Omnis server, you need both the standard Web server plugin, and the RESTful Web server plugin to be installed with the Web server, i.e. you need to install both omnisapi.dll and omnisrestisapi.dll.

The client scripts automatically generate a URL for push by converting the parameters in the web page. For example, if your HTML page for the JavaScript client uses the URL:

```
http://localhost:8080/omnisservlet
```

then the client scripts will convert this to:

```
http://localhost:8080/omnisrestservlet
```

for the push connection. When generating push URL, Omnis only amends the plugin name part of the complete URL, so in the above case, omnisservlet becomes omnis**rest**servlet. You can see the URL used for push connections by using browser debugging tools.

If you are not using standard names in your HTML page, there is a parameter in the Omnis configuration file (config.json) that allows you to override the default push URL generated by the scripts: this can only be used when using openpush in a server method. To configure this set the member "overridePushURL" of the "server" entry to the desired URL.

Remote Form Method

To support push connections there is a method for remote form instances called $pushdata(), which has the following syntax:

- **$pushdata(**wRow[~&cErrorText]**)**
  Used with $clientcommand openpush. The method pushes the row wRow to the client which results in a call to the client-executed method $pushed in the remote form instance on the client, passing wRow as the parameter. wRow must be JSON compatible, so it can only contain simple types: character, boolean, integer, number, date, list and row.

Omnis maintains a queue of pushed data for the remote task, which is independent of calls to openpush. As soon as a push connection arrives from the client, Omnis sends all queued pushed data that the client has not yet received as the response. The client then processes the response, and issues a new push connection to the server, telling the server it has received the data. This allows the server to remove the received data items from its queue, and free their memory. Typically, at this point there will be no more queued data. The connection stays open, and as soon as the server code calls $pushdata, Omnis sends the data as the response to the client.

This gives the impression of a permanent pipe from the server back to the client, with acknowledgement of pushed data received by the client, so pushed data should not go missing.

Typically, you would take data from the row returned by $pushdata and assign it to an instance variable, or subset of variables, to update the remote form.

There is a tech note TNWS0005 to show how you would use a RESTful web service with the openpush client command in the JavaScript Client.

Poll delay

The ‘openpush’ client command has an optional column which can be passed in its parameter row. The ‘maxPollDelay’ column allows you to override the default maximum delay (1000ms) between the client receiving a ‘$pushdata()’ from Omnis, and making a new connection to Omnis ready for the next ‘$pushdata()’ command. Passing a value of 0 (or less) will not change the maximum delay.

If your application bounces back and forth between client & server in quick succession (you call a server method from $pushed, which in turn calls $pushdata), you may find that reducing this makes your application more responsive. There is a small overhead to reducing this too low, however, so it’s recommended to leave the default value unless you have a need to change it.

Remote Objects

Remote Object classes (or Remote Objects) are Object classes that are instantiated and executed entirely on the client, in the JavaScript Client. Each Remote Object class instance has a JavaScript object “class” that directly corresponds to it on the client.

Remote Objects are useful if you have some code that you want to be executed purely on the client, and you want to use it in multiple remote forms, so a Remote Object would provide a good way to structure the code in your application, that is, it provides an alternative to having to inherit methods from a Remote Form superclass, so may be useful in a serverless-client based mobile app.

Creating Remote Objects

The Studio Browser window allows you to create a new **Remote Object,** create a subclass of an existing remote object class, and edit a remote object. Editing a remote object opens the Method Editor, in the same way as when you edit a normal object class. Within the Method Editor itself, the main difference is that every method in a remote object class is *always marked as client-executed*.

The JSON library representation now includes support for remote objects. You can print methods in a remote object class.

Find and replace supports remote object classes, and has an additional entry in the class selection menu, to select remote object classes.

The inheritance tree includes a node for remote object classes.

Omnis Language

Library Notation

The notation for manipulating remote objects is similar to that for objects. There is a new group within each library, called $remoteobjects, containing all of the remote object classes in the library. Each remote object class has a subset of the properties and methods supported by object classes:

Variables

Remote object classes can have class and instance variables. These are restricted to the set of client execution data types: var, date, list, row, and (new for remote object support) object. In addition, each method in a remote object class can have local variables, which are likewise restricted to the set of client execution data types including object.

Creating Instances

You create an instance of a remote object by specifying the remote object class name as the subtype of a variable in a remote object (see the previous section) or for remote forms, either:

- a local variable of type object in a client-executed method

- or a remote form instance variable of type object.

Note that this means that remote form instance variables of type object can now have a remote object class as their subtype, in addition to an object class or a non-visual external object. Remote form object variables with a remote object as their subtype are not synchronised between client and server – they exist only on the client.

Behavior

You can write the methods in a remote object class in the same way as you create the methods in an object class, except you are restricted to client-executable code. Inheritance works as you would expect using the normal Omnis mechanism, although you cannot override variables in a subclass – you must inherit superclass variables. Variables are referenced as you would expect, e.g. you can just use iName or you can use $cinst.iName.

However, note that you cannot use $new to create a new remote object instance. This is because the Omnis server needs to be able to quickly parse a remote form and its superclasses in order to determine the remote object classes it uses, in order to generate the code correctly.

Remote objects do not execute $destruct, because they are JavaScript objects (which are naturally garbage collected by the execution environment).

If you execute a remote form method marked as client-executed, by calling it from a server method, then because the method is actually executing on the server, Omnis will generate an error if you try to use a remote object.

When coding in the Method Editor, the Code Assistant will only provide assistance for remote object instance variables, and object instance variables, when you are coding for an environment that is applicable: so for example, you would get no assistance for a remote object instance variable when coding a server-executed method.

When passing remote objects around between methods, bear in mind that they are passed by reference, so they are never copied.

$cwind for remote objects

You can use the notation $cwind from code written in a remote object, to refer to the top-level remote form instance that contains the remote object, for example, you can write code like the following in a remote object method:

```
Calculate $cwind.$objs.[pName].$backcolor as pick($cinst.$isodd(),kMagenta,kCyan)
```

In addition, you can use the notation $cinst.$container in a remote object to refer to the remote form that immediately contains the remote object.

Code Generation

The Omnis server automatically generates the JavaScript code for remote objects, in a similar way to how it generates JavaScript code for client-executed methods in remote forms. The JavaScript for each remote form contains the JavaScript for all of the remote objects it uses, using a conditional test which means that if 2 remote forms use the same remote object, the code used for all instances of the remote object will be that loaded with the first remote form.

If you modify and save a remote object class, Omnis will regenerate the code when the remote form is re-loaded.

Remote Form Properties

Like other types of Omnis class, **Remote Form** classes *have many properties that control their appearance and behavior*. You can set most properties of a remote form class in the Property Manager, but the properties for remote form instances must be assigned or controlled in the code in your app. Remote form instances also have methods: see the Remote Form Methods section.

Many of the properties, including many of the standard form properties, are reasonably self-explanatory, but some properties that are specific to remote forms require further explanation and are described in the following sections. The $serverlessclient property enables the *Standalone Mobile Apps* capability, and is described later in the Serverless Client section, while the $stringtabledata and $stringtabledesignform are discussed in the Localization chapter.

Responsive Forms

Responsive design is a technique used to design form layouts that cater to different devices or screen sizes, including phones, tablets, and desktops, from a single remote form class. The motivation for employing responsive design is to create *a single form,* with one set of code methods, that *adapts its layout automatically* when it is displayed on a range of different devices, or when the client browser is resized. For standard web pages, responsive design is implemented using CSS media queries and breakpoints, and Omnis takes a similar approach by allowing you to specify a number of ***layout breakpoints*** in a single JavaScript remote form, where each breakpoint corresponds to a *different layout* for the fields and other controls on your remote form.

Migrating to Responsive Form design

All new remote forms created in the Studio Browser via the New Class option or the Remote Form wizards are set to the responsive layout type by default.

Remote forms in converted libraries (Studio 8.0 or earlier) will continue to use the $screensize property to specify the layout for different devices. There is a migration tool, available under the **Tools>>Add-Ons** menu, **JS to Responsive** option, that allows you to migrate existing $screensize based remote forms to the responsive type. (Note the old Sync Screen tool only applies to the old $screensize based remote forms, and should not be used for responsive forms.) See Remote Form Migration.

Form Layout Type

JavaScript remote forms have a property, **$layouttype,** that specifies how the layout of the form is designed: it can be set in design mode to one of the kLayoutType... constants, and *is only assignable when the remote form is empty, when it does not contain any controls*. You can return the value of $layouttype in a remote form instance, but you cannot change it at runtime in your code.

The possible values for $layouttype are:

- **kLayoutTypeResponsive**
  The remote form has a responsive layout with layout breakpoints, as specified in the form toolbar and stored in the $layoutbreakpoints property as a comma-separated list. A remote form can have a different layout for the fields and other controls for each breakpoint value. All new remote forms are set to this type.
- **kLayoutTypeScreen**
  This option corresponds to remote forms in Studio 8.0 or earlier libraries, and uses the old $screensize property containing a number of fixed screen sizes. An existing remote form in a library converted from Studio 8.0, or earlier, will be set to this layout type (you can use the migration tool under the Tools>>Add-ons menu to convert a $screensize based form to responsive: see Remote Form Migration).
- **kLayoutTypeSingle**
  The remote form has a single layout. This type could be used for applications intended to be deployed on desktop web browsers only: you can use the $edgefloat property for controls to resize or reposition them when the browser window is resized (this layout type does not allow breakpoints to be set).

A responsive remote form does not have the following properties, since they are not relevant to responsive design: $resizemode, $screensize, $width, $height, $horzscroll, or $vertscroll, however $edgefloat still applies to components in responsive forms.

If you change $layouttype to kLayoutTypeSingle, and the $resizemode property is set to kJSformResizeModeNone, then $resizemode will be set to kJSformResizeModeFull automatically to make it resizble.

Creating Responsive Remote Forms

You can create a new Responsive Remote form class in the Studio Browser using the **New Class>>Remote Form** option, and in this case, the $layouttype property is set to kLayoutTypeResponsive automatically. The remote form wizards, available under the **Class Wizard** option in the Studio Browser, also create remote forms with the responsive layout type. If you want to change the layout type, you must change it *before you add any controls,* since you cannot change the form layout type once it contains any controls.

A new responsive remote form contains two layout breakpoints by default: these are set to **320** and **768** which correspond to the relative widths for mobile devices and desktop computers (note the default breakpoint values for new remote forms are set in the $initiallayoutbreakpoints library preference).

Remote forms can have only one breakpoint, but in most cases, you would define two breakpoints to cater to mobile devices and desktops or tablets.

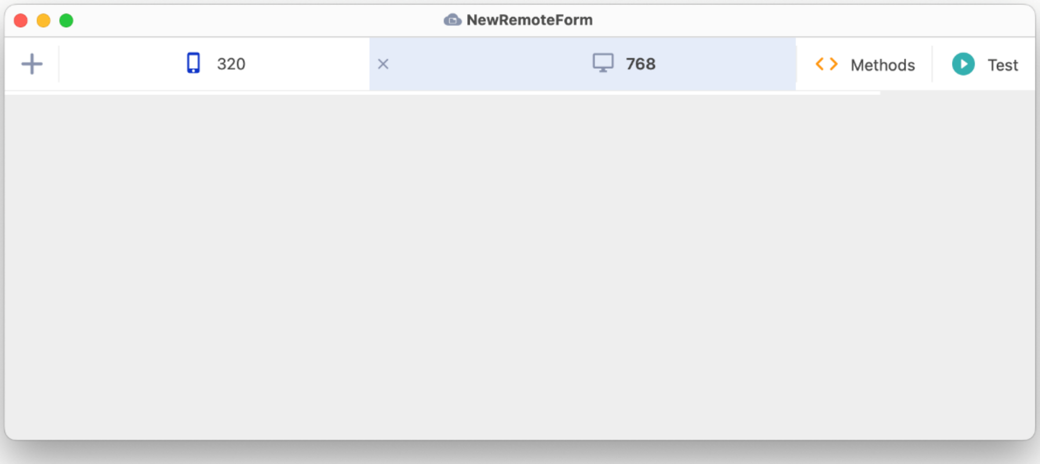


Figure 89:

Changing and Adding Breakpoints

You can change the default breakpoints to suit the layouts you wish to support in your application. You may find, for example, that setting two breakpoints is enough to cater to mobiles and tablets or desktop screens, and then use the floating edge properties of objects ($edgefloat) to resize and reposition them for different device or screen sizes (the remote form wizards take this approach).

Each layout breakpoint must be a positive integer in the range 100 to 32000, with at least 32 pixels between any breakpoints; therefore, you cannot create a breakpoint with an existing value, or within 32 pixels of an existing breakpoint. The minimum width of the first breakpoint is 100.

The Breakpoints for a Remote form are shown in the Design bar. Clicking on a layout breakpoint makes it the current layout.

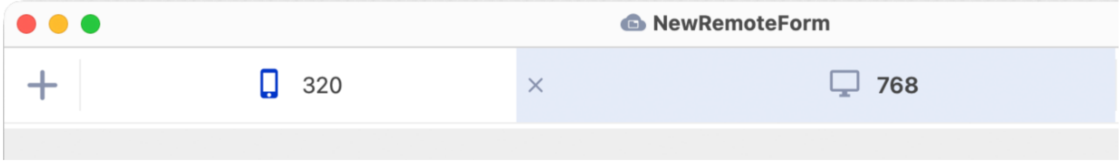


Figure 90:

You can change, delete or add new layout breakpoints using the toolbar at the top of the remote form design screen, as follows:

- To **change** the value of a layout breakpoint, you can *drag the right edge* of the current breakpoint in the toolbar, or you can *double-click* on the number in the form toolbar and enter a new value, or press **Ctrl/Cmnd-E** to edit the value.
- To **delete** a breakpoint, click on the Delete (X) button when the breakpoint is selected, or press **Ctrl/Cmnd-D** when the breakpoint is selected (the delete button is not shown *when there is only one breakpoint*, since this is the minimum number of breakpoints for a responsive form).
- To **add a new** layout breakpoint, click on the '+' button in the top-left corner of the form toolbar, or press **Ctrl/Cmnd-L** when the remote form is selected, and enter a breakpoint value.

You can right-click on a breakpoint (which also makes it the current breakpoint) to open a context menu which provides options to edit the breakpoint value and delete the breakpoint.

When adding a new layout breakpoint, all breakpoint-specific properties are copied from the nearest breakpoint, including the size and position coordinates of the components in the existing breakpoint, plus the following: layout padding for the form, edge float, align, drag border, error text pos, and 'visible in breakpoint' properties.

Deleting Breakpoints

When you delete a breakpoint, the positioning and individual properties you have set *for all of the fields and controls in the layout are lost,* so use this option with caution. You can restore a deleted layout breakpoint immediately after deleting it using the **Undo** option. If undo is not available, you will lose the breakpoint and any custom settings for the all the fields and controls in that layout; in this case, you would have to recreate the layout again.

Layout Breakpoints

A responsive remote form must have *one or more* layout breakpoints. Layout breakpoints are widths measured in CSS pixels, so they represent logical sizes rather than physical sizes. The JavaScript client chooses the layout for one of the breakpoints defined in the form based on the logical width of the area in which the remote form is to be displayed in the browser on the device.

- For a **desktop browser,** the width would be the width of the browser window (which can be resized), although note that responsiveness also applies to remote forms displayed in a subform control or subform set (in which case the width is the width of the subform control or container for the subform set).
- For a **mobile device,** the width is most likely to be the width of the device screen itself, although again, a form on a mobile device can be loaded in a subform control or subform set which may be narrower than the device screen.

The client chooses the most appropriate layout for the device, from all the layouts available in the form. Specifically, the client uses the layout for the largest breakpoint that is less than or equal to the display area width, or if no such breakpoint exists (because all breakpoint widths are greater than the display area width), the layout for the smallest breakpoint.

Once the client has chosen a breakpoint, the client will apply floating and component properties to make use of the available extra width (if any), and if there is no extra width, the client will automatically turn on horizontal scrolling if necessary.

Layout Breakpoint Properties

Remote forms have a property called $layoutbreakpoints, which stores the layout breakpoints for a remote form. This is a comma-separated list of one or more breakpoint values, and these values are shown and edited in the toolbar in the remote form design screen: you cannot set layout breakpoints for a form in the Property Manager. You can return the value of $layouttype in a remote form instance, but you cannot set it at runtime.

When you create a new responsive remote form, the layout breakpoints in the form (and the value of $layoutbreakpoints) are initialized with the value of the library preference $initiallayoutbreakpoints. If you wish to create new remote forms with different layout breakpoints you can edit this preference: to do this, select the library in the Studio Browser and set the property under the Prefs tab in the Property Manager.

A responsive remote form has a property, $currentlayoutbreakpoint which is the value of the current layout breakpoint. In design mode, the current breakpoint is highlighted in the form toolbar: it is not shown in the Property Manager. At runtime, the value of $currentlayoutbreakpoint may change if the end user resizes their browser window, or changes the orientation of a mobile device.

Minimum Layout Height and Padding

Each layout breakpoint in a remote form has a property **$layoutminheight,** which is the minimum height of the responsive layout. The default setting of $layoutminheight is zero which means the minimum height of the form is set to the bottom-most coordinate of all controls plus an additional 2 pixels for padding (other non-zero values must be in the range 100 to 32000 inclusive). The minimum height is indicated in design mode as the white area containing all the controls; the surrounding area in the form design screen is shaded gray. When the available client height at runtime is larger than this value, controls can float to use the additional vertical space, depending on their $edgefloat properties.

The **$layoutpadding** property allows you to set the amount of padding under the bottom-most control on the form. By default, the bottom edge of the form is set to 2 pixels under the bottom-most control.

The range for $layoutpadding is 0 to 512 which is added to the bottom-most coordinate of all controls, to generate the minimum layout height when $layoutminheight is zero. When available client height is larger than this, the controls on the form can float. A value is stored for each breakpoint.

When you create a new remote form class (or convert an existing remote form), $layoutpadding is set to 2 by default for each breakpoint. The default value of $layoutpadding is specified in the “responsiveLayoutPadding” item in the “defaults” section of the Omnis configuraration file (config.json), which is set to 2 by default.

When a remote form is accessed for the first time, e.g. in a converted library, the value of $layoutpadding is initialized to the default padding (unless the remote form is read-only, in which case the default value is used, but not written to the class).

Layout Breakpoints for Subforms

Subforms within the inheritance hierarchy of a set of responsive remote forms *do not have to have the same layout breakpoints*. Therefore, a subform can have different layout breakpoints to its superclass.

What breakpoints should I use?

In general, you need to create a breakpoint for the smallest device within each category of device you wish to support (phone, tablet, or desktop). Therefore, the value of the first breakpoint would be the logical width of the smallest phone you wish to support (bearing in mind logical dimensions are not the same as the pixel dimensions, which depend on the density of the screen). For example, the logical dimensions of the iPhone 13/14 are 390 x 844 (Pro versions are larger), and the Samsung Galaxy S21 is 360 x 800, so you could set the first layout breakpoint to 350 to allow a safe margin and to accommodate form layouts for both phones; or you could retain the default 320 breakpoint to cater to older phones that have a smaller logical width.

Similarly, to set the layout breakpoint for tablets you should consider the minimum width for the range of tablets you wish to support. The default breakpoints defined in a new remote form (320 and 768) provide support for a wide range of mobile devices or tablets, both in vertical and horizontal orientations, but you may need to adjust the default breakpoints to suit your requirements, or as new phones are released.

Adding Controls

When you add a control to a responsive remote form it is added to the current layout and all other layout breakpoints: initially, a control will be in *the same position in all layouts,* but you can switch to another layout and change its position and other object properties for that layout, such as $edgefloat. If you delete a control from one layout it will be removed from all other layouts, and any individual object property settings will be lost.

Copying Layouts

You can copy the layout from another layout to the current layout using the **Copy Layout from Breakpoint** option in the remote form. To do this, select the layout breakpoint you want to update, right-click on the background of the form, select the **Copy Layout from Breakpoint** option, and choose the breakpoint value of the layout you want to copy from (values other than the current breakpoint are shown). This has the effect of synchronizing the layouts of the current and selected breakpoints, by applying the size and position properties of all components in the chosen layout, including their $edgefloat settings. (Note this has a similar function to the Sync Screen tool available for old $screensize based forms.)

The **Copy Layout from Breakpoint** menu option can also apply to selected objects only if an object or multiple objects are selected. The layout properties of the selected objects in different breakpoints are set to the same values, while the other non-selected objects are unaffected; in this case, the menu option text changes to show ‘selected fields only’.

Assigning Properties

The ‘Copy <property> To All Other Layout Breakpoints’ option on the Property Manager context menu allows you to copy the property value of a control to other instances of the control on all other breakpoints.

In addition, the ‘Copy Position To All Other Layout Breakpoints’ option allows you to copy the position (meaning left, top, width, height and edgefloat) of a control to all other breakpoints.

Control Size and Layout Properties

The following layout properties are stored for *each control* for *each layout breakpoint,* that is, they can be set to different values for each layout: $left, $top, $width, $height, $align, $edgefloat, $dragborder, $errortextpos, and $visibleinbreakpoint, which allows you to hide a control for certain layouts. For example, you could use this property to show a vertical tabbar for one layout and a horizontal tabbar for another layout.

When setting the $align, $edgefloat, $dragborder, $errortextpos and $visibleinbreakpoint properties in the Property Manager, you can assign the selected value to the control *on all layouts* by checking the ‘Set for all layout breakpoints’ option in the property droplist.

Responsive Form Methods

Remote form classes have a number of methods to allow you to manipulate the layout breakpoints in the form (note these cannot be used in remote form instances, since you cannot change breakpoints at runtime):

- **$addlayoutbreakpoint**(iBreakpoint[,&cErrorText])
  Adds a new layout breakpoint to the responsive remote form at position iBreakpoint. Returns true for success, or false and cErrorText if an error occurs

- **$movelayoutbreakpoint**(iOldBreakpoint,iNewBreakpoint[,&cErrorText])
  Moves breakpoint iOldBreakpoint for the responsive remote form to iNewBreakpoint. Returns true for success, or false and cErrorText if an error occurs

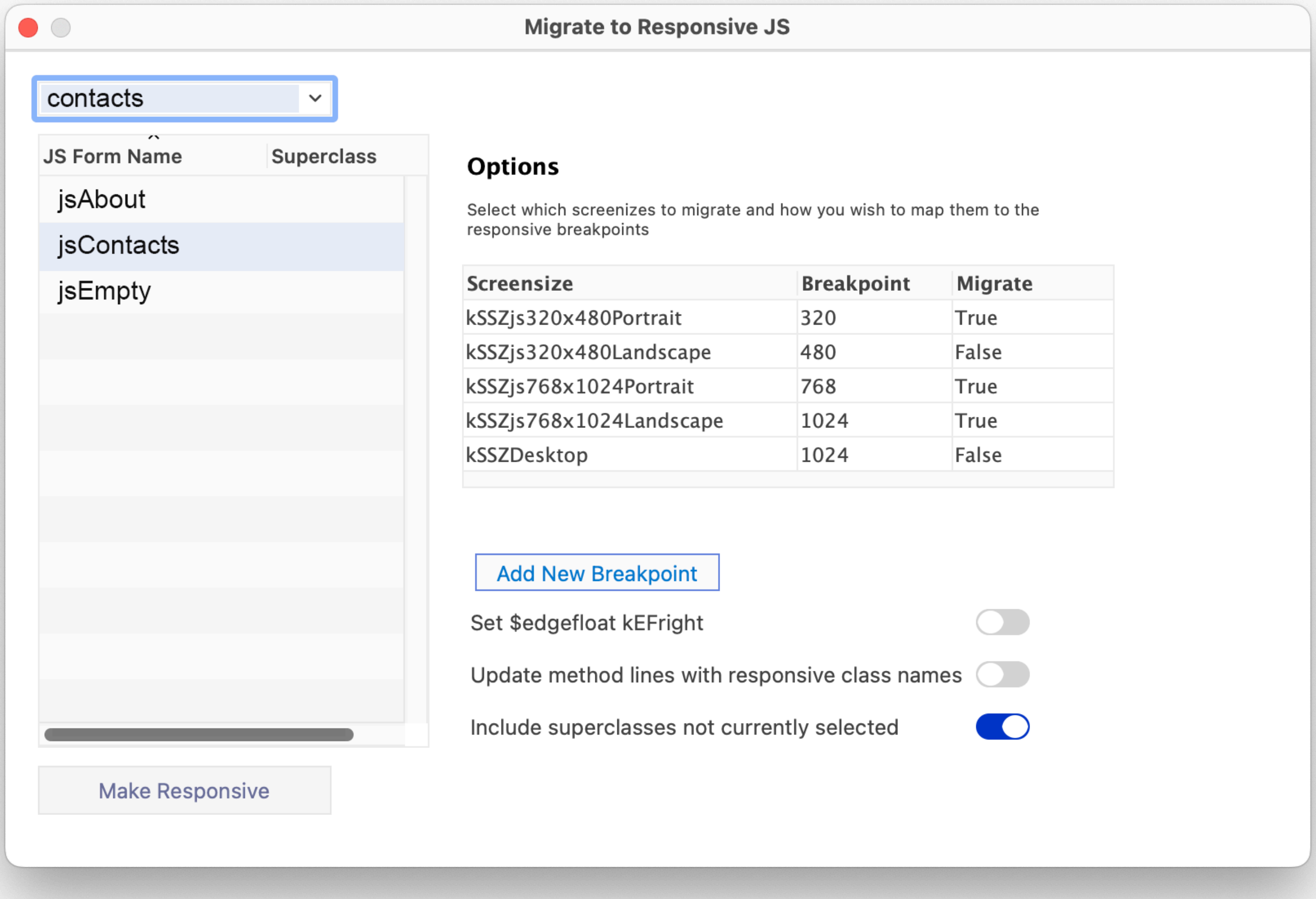
- **$deletelayoutbreakpoint**(iBreakpoint[,&cErrorText])
  Deletes the layout breakpoint at position iBreakpoint from the responsive remote form. Returns true for success, or false and cErrorText if an error occurs

Remote Form Inheritance

$layouttype cannot be overridden or changed in a subclass. $layoutbreakpoints cannot be inherited: each class has its own set of layout breakpoints. However, $layoutminheight can be overridden.

Remote Form Migration

The **Remote Form Migration** tool, under the **Tools>>Add-Ons** menu **JS to Responsive** option, allows you to convert an existing JavaScript remote form in a library converted from Studio 8.0, or earlier, to the responsive form type. The migration tool creates new layout breakpoints corresponding to the old screen sizes available in remote forms in previous versions, and tries to adjust the positioning and layout of fields to fit those breakpoints. The migration tool creates a new responsive remote form with breakpoints and modified screen layouts, based on an existing remote form, and retains the old unmodified form in your library.



Migrate to Responsive JS

contacts

JS Form Name	Superclass
jsAbout	
jsContacts	
jsEmpty	

Options

Select which screenizes to migrate and how you wish to map them to the responsive breakpoints

Screensize	Breakpoint	Migrate
kSSZjs320x480Portrait	320	True
kSSZjs320x480Landscape	480	False
kSSZjs768x1024Portrait	768	True
kSSZjs768x1024Landscape	1024	True
kSSZDesktop	1024	False

Set $edgefloat kEFright ○

Update method lines with responsive class names ○

Include superclasses not currently selected ●

Figure 91:

The migration tool will create breakpoints at 320, 768, and 1024 in the new remote form, and assign them to the form layouts corresponding to the old screen sizes (the kSSZ… constants) set under $screensize: to create a breakpoint it must be set to True in the **Migrate** column in the Migration tool window. The 480 breakpoint is available but is not enabled by default, since it is not needed in the new responsive form.

You can add a new breakpoint using the **Add New Breakpoint** button and assign that value to one of the old screen sizes; the new Breakpoint value is added to the dropdown menu in the Breakpoint column. For example, you may wish to create a breakpoint at 300 and assign it to the old phone screen size (320x480) to ensure that all content is displayed on all types of phones.

The **Set $edgefloat kEFright** option sets the $edgefloat property of certain controls to kEFright to ensure that when the form is resized in the browser the right edge of those controls is also resized or moved. In this case, only controls with no other controls to their right, which are generally on the right-hand side of your form, are updated. Specifically, the $edgefloat property of any buttons is set to kEFleftRight, rather than kEFright, to ensure they float without resizing when the browser window is resized.

The **Update method lines…** option will replace all references in your code to the old remote form name to the new name, so your code continues to work.

When you have set up the appropriate options you can click the **Make Responsive** button to create the new responsive form(s), which are placed in a new folder in your library. You can modify them, or test them straight away using Ctrl/Cmnd-T.

Migration Log and detecting form width

When you have run the migration process, the tool creates a change log which will contain any issues that may need your attention. This may include any places in your code that use the old $screensize constants (kSSZjs…), which no longer apply to responsive forms.

Screen Type Layout (kLayoutTypeScreen)

The following section refers to Remote forms when the $layouttype is set to kLayoutTypeScreen. This layout type enables the use of the old $screensize property and fixed screen sizes, available in Studio 8.0 or earlier, which you are advised not to use for new remote forms. Any remote forms in an existing library which is converted to Studio 8.0 or earlier will have this layout type. Note that in addition, only remote forms of type kLayoutTypeScreen trigger the evLayoutChanged event when their layout changes (from Studio 10.0).

The $screensize property provides a number of fixed screen sizes for displaying remote forms on desktop browsers, tablets and phones. As with responsive remote forms, each fixed screen size uses the same set of objects (and methods) and the remote form class stores the position of the fields *for each screen size* setting.

The following fixed screen sizes and orientations are available (for $screensize based forms only, not responsive type forms):

- **kSSZDesktop**
  for remote forms running in desktop browsers; in effect, the screen size is unspecified and the $height and $width of the remote form *in design mode* is used to size the form in the browser
- **kSSZjs320x480Portrait** or **kSSZjs320x480Landscape**
  For mobile devices with screens 320 x 480 px (at 96dpi) in Portrait/ Landscape orientation
- **kSSZjs768x1024Portrait** or **kSSZjs768x1024Landscape**
  For tablets with screens 768 x 1024 px (at 96dpi) in Portrait/ Landscape orientation

When opening (constructing) a remote form, the JavaScript Client uses the most appropriate fixed screen size and orientation stored with the form, for the screen size and orientation of the current device. If the user swaps from portrait to landscape, or back again, the JavaScript Client repositions the controls automatically.

Form Width and Height

When specifying the width and height for the mobile fixed screen sizes, you can set the $width and $height properties to match the exact coordinates in the current setting of $screensize, allowing for the mobile title bar which is 20 pixels high. If you have set $designshowmobiletitle to kFalse you may want to add 20 pixels to the height of the form.

Adding new fixed screen sizes

The screen sizes enabled in the $designedscreensizes library preference will be used to populate the $screensize property in the Property Manager. The omnisobject containing the JavaScript client in the HTML page has the 'data-dss' attribute, which contains the designed screen sizes for the library. If you use forms from more than one library in a single client instance, each library must have the same set of $designedscreensizes. If not, a runtime error will occur when trying to use a form from another library.

If you change the screen sizes supported in the $designedscreensizes library preference, all the HTML files for all remote forms in your library need to be rebuilt to reflect the new set of screen sizes: this is done automatically when you test a remote form since the HTML file is rebuilt every time you test a remote form. Note the jsctempl.htm template file contains the data-dss attribute and any screen sizes currently implemented for the JavaScript client.

Testing Form Layouts in Firefox

If you are using Firefox during development, you can test different layouts for mobile and tablet screen sizes in a single browser window using the 'Responsive Design View' mode: note this is a feature of Firefox and is not available in other browsers. This may save you a lot of time during the initial stages of designing your mobile application, since this avoids having to test your app on multiple devices to test different sizes and layouts. However, we recommend that you should test your final app on any real device that you wish to support when you are ready to deploy your app.

To enable this functionality, you need to set the 'gResponsiveDesign' flag to true in the 'ssz.js' script file located in the html/scripts folder in your Omnis development tree. For this to take effect, you must restart Omnis after setting the responsive design flag. To enable this mode in Firefox, go to the Tools>Web Developer menu option and select 'Responsive Design View': you will need to show the Menu bar in Firefox to see this option. Then when you test your remote form in Firefox, you can select different screen sizes and orientations in the dropdown menu in the Firefox browser window, and your remote form will redraw using the appropriate screen size specified in $screensize for the remote form. When you have finished testing using this mode, you should set gResponsiveDesign in the 'ssz.js' script file back to false.

Sync Screens Tool

The Sync Screens tool applies to $screensize based forms only; it does not apply to responsive forms.

The **Sync Screens** Tool configures the components on the different fixed screen sizes stored in a single remote form. The **Sync Screens** tool is available under the **Tools>>Add Ons** menu in the main Omnis menu bar.

To use the Sync Screen tool you need to select a library from the Library dropdown and then select the JavaScript form in which you wish to synchronize objects. The 'Source Screensize' is used as the starting point upon which the other screen sizes/layouts are based (the desktop size/layout is chosen by default). You can choose which screen sizes/layouts will be synchronized, and under the 'Options' check boxes whether or not to scale objects by horizontal or vertical position and/or by width and height. If you don't want a particular object to be resized or repositioned by the tool, you can lock it in the remote form in design mode (Right-click the object and select Lock) and enable the 'Ignore Locked Components' option (enabled by default). When you have adjusted the settings, click on the **Sync** button.

You should change the setting of $screensize in your remote form and check the layout of the objects for each screen size/layout. You should also test the form in a browser and on different devices to check that the form objects have been sized and positioned correctly.

Resize Mode (Screen & Single layout only)

Remote forms with the layout type kLayoutTypeScreen and kLayoutTypeSingle have the $resizemode property which allows a form displayed in a desktop web browser to be resized ($resizemode is not available for kLayoutTypeResponsive forms). The $dragborder and $edgefloat properties can then be used to allow JavaScript forms and components to be dynamically resizable at runtime in the end user's browser.

The $resizemode property only applies when the remote form is being displayed in a standard browser window on a desktop computer or laptop, that is, the property does not apply when the form is displayed in a browser on a mobile device or when the form is being used as a subform since in this context the form size is fixed.

The $resizemode property determines whether or not a form resizes when the end user resizes the browser window. The value of $resizemode is one of the kJSformResizeMode… constants that specify how the form behaves when it initially opens and when the browser window is resized. The kJSformResizeMode… constants are as follows:

- **kJSformResizeModeNone**
  The form does not change size when the browser window is resized and the form is positioned at the left of the browser window. This corresponds to the behaviour in Omnis Studio 5.2.x

- **kJSformResizeModeCenter**
  The form does not change size when the browser window is resized but the form is centred horizontally in the browser window, only if its width is less than the browser window width
- **kJSformResizeModeAspect**
  The form resizes itself as the browser window is resized maintaining its aspect ratio to fit the browser window; it will not resize to a size smaller than the designed size in the remote form class
- **kJSformResizeModeFull**
  The form resizes itself to fit the browser window, regardless of aspect ratio; it will not resize to a size smaller than the designed size in the remote form class

You can assign $width and $height of the remote form at runtime, however this may conflict with $resizemode, so you should only assign these properties when $resizemode is kJSformResizeModeNone.

Component Transitions

Remote forms have a property, **$animatelayouttransitions,** which specifies whether or not the controls on the form will animate to their new position and size when the form layout or orientation changes on the client. If this property is set to kTrue, all the controls on the form will animate on the transistion, e.g. when changing from vertical to horizontal orientation, or when resizing your desktop browser window. You can stop the animation for individual controls by setting the **$preventlayoutanimation** property to true for the control. (The transition properties apply to responsive remote forms and the old $screensize based forms.)

The animation time is hard-coded to 500ms, but you can override this for individual controls using JavaScript as follows:

```
Calculate lControl as $cinst.$objs.myButton1
JavaScript:lControl.animateLayoutTime = 1000;
# Set layout transition animation time to 1000ms for myButton1
```

Or, to set a new animation time for ALL controls on the form, execute the following in the remote form’s $init method:

```
JavaScript:ctrl.prototype.animateLayoutTime = 1000;
```

Initial Field and Tabbing Order

The $startfield property specifies which field in a remote form will get the focus when the form is opened; it takes the field number of the control as specified in the $order property of the control. The relative values of the $order property for all the controls in your remote form determines the tabbing order of the controls in the form. In general useability and accessibility practices it is usual to specify the left- or top-most control of the form as $order = 1 and follow consecutive order number values across to the right (if you have rows of controls) and downwards.

For inherited forms, the $inheritedorder property determines the tabbing order for the first inherited field: zero means maintain the designed order from the base class through to this class.

Responding to OK and Cancel Keys

You can specify which object in the remote form receive OK and Cancel events from the keyboard, e.g. when the end user presses the OK or Cancel button. For the JavaScript remote forms, $okkeyobject specifies the object in the form that receives evClick when the user presses Return or Enter. Similarly, $cancelkeyobject specifies the object that receives evClick when the user presses Escape. Both of these properties only apply if the field currently with the focus does not use the key that is pressed, in which case these properties will have no effect.

Form background and Subforms

The $okkeyobject and $cancelkeyobject properties are activated when the focus is on the containing form, so $okkeyobject and $cancelkeyobject will now receive a click when the Enter or Esc keys are pressed, and the focus is on the whitespace within its container. Each parent form (when working with subforms) will be checked for an $okkeyobject or $cancelkeyobject until it reaches the top form. The exception to this is if the subform is contained within a subform set, and in this case, it will keep checking parent forms until it reaches its containing subform.

Form and Component Transparency

Remote forms have the $alpha property which sets the transparency of the form (an integer from 0 to 255, with 0 being completely transparent and 255 opaque). In addition, $backalpha lets you control whether or not subforms in the main form use the background color of the subform field or the form itself. The majority of the JavaScript components have the $alpha and $backalpha properties which control the transparency of the foreground and background colors of the component.

In combination with the animation methods, you can use the $alpha of a form or control to make elements in your form appear and disappear. The About windows in the sample apps (available in the Welcome screen when you start Omnis) are displayed by setting the $alpha property of the About subform and using the animation “ease in” effects. See the Animations section.

Gradients

To create a gradient for the background of a JavaScript remote form, you can select a gradient fill pattern for $backpattern and control the start and end colors for the gradient by setting $forecolor and $backcolor.

Remote Form Instances and Methods

When you “open” or test a remote form in Omnis it is opened in a web browser. In development mode, this will be the default web browser on your development computer, but when your app is deployed, the remote form will open in the end user’s web browser, or the browser on a mobile device, or within a wrapper application for standalone mobile apps. When a remote form is opened a *Remote Form Instance* is created which will have a number of *methods* that you can use in your code to perform various actions.

Remote Form Instance Properties

Remote form instances have a number of properties, including:

- **$remotemenu**
  which is the current remote menu instance. This is only set when evOpenContextMenu is being processed: see the Remote Menus section
- **$sqlobject**
  is the JavaScript Client SQL Object which is only available in client-executed methods running in a wrapper application: see the SQL Object section
- **$layouttype**
  returns the current layout type of the remote form instance; note you cannot set it in your code

Remote Form Methods

Remote form instances have the following methods, to enable animations, client messages, client commands, and the ability to open a web page from within the client:

- **$beginanimations()** and **$commitanimations()**
  $beginanimations(iDuration [, iCurve=kJSAnimationCurveEaseInOut]) After calling this, assignments to some properties are animated for iDuration milliseconds by $commitanimations(); the $commitanimations() method animates the relevant property changes that have occurred after the matching call to $beginanimations(); see the Animations section
- **$clientcommand()**
  $clientcommand(cCommand, wRow) Executes the command cCommand on the client device using the parameters in the row variable wRow; see the Client Commands section for possible client commands.
- **$maximize()** and **$minimize()**
  For subforms in a subform set only: maximizes or minimizes the remote form instance if it is a member of a subform set
- **$setcurfield()**
  $setcurfield(vNameOrIdentOrItemref [,bSelect=kFalse]) sets the current field on the client and places the focus in the field; for mobile devices the soft keypad may be initiated (depends on the OS); if bSelect=kTrue, and if supported by the control, all of its content will be selected; executing $setcurfield('') will remove the focus from the current field; you can specify the field by name, ident, or item reference

- **$showmessage()**
  $showmessage(cMessage[,cTitle]) displays an OK message on the client machine using the specified cMessage and cTitle; multiple calls to $showmessage() during the processing of a single event will result in only the last call to $showmessage() having any effect and that message being shown; see Client Messages
- **$loadfinished()**
  is a client-executed method that allows you to check when all subforms of a form have been loaded; it is called after all the subforms that belong to the parent remote form instance have finished loading and their $init methods have been called, so you could create a client method called $loadfinished to perform any actions you want after all subforms have loaded
- **$showurl()**
  $showurl(cURL[,cFrame,cWindowProperties,cWindowRef]) opens the URL in a new window or frame on the client machine; cURL specifies the URL of the HTML page; cFrame specifies the HTML frame name; if cFrame is empty, the page is displayed in a new window, otherwise it is displayed in the specified frame of the current window
- **$closeurl()**
  $close(cWindowRef) closes a browser window that was previously opened by the $showurl() method, takes a single parameter, a string identifier to the window, returned in the fourth parameter of the $showurl() method

See also Client Methods for remote form methods that can be executed on the client (including $init method).

The cWindowProperties parameter for the $showurl() method is ignored if cFrame is not empty. Otherwise, it has the same format as the JavaScript argument to 'window.open', for example, "toolbar=0,menubar=1" specifies that the browser window will have a menubar, but not a toolbar. The keywords are all boolean (0 or 1) except for the width, height, top and left, which are numbers in pixel units. Possible keywords are:

Keyword	Description
toolbar	specifies if the browser window has a toolbar
status	specifies if the browser window has a status bar
menubar	specifies if the browser window has a menu bar
scrollbars	specifies if the browser window has scrollbars
resizable	specifies if the browser window is resizable
location	specifies whether the browser window has a location bar
directories	specifies whether the browser window displays Web directories
width	width of browser window
height	height of browser window
top	top coordinate of browser window
left	left coordinate of browser window

$redraw and $senddata Methods

The $redraw and $senddata Methods are only relevant to remote forms used with the now obsolete Web Client plug-in, so do not apply to JavaScript based remote forms. Redraws are handled automatically for JavaScript remote forms and controls, so the $redraw() method is not required and if it is called it does nothing. In addition, the $senddata() method was used in the old plug-in to control when data was returned to the client, but this is irrelevant for JavaScript Client which handles the transfer of data between the client and server automatically.

Client Messages

The $showmessage() method allows you to display a message on the client device. This method only applies to remote tasks that are associated with remote forms, that is, the method does not work for remote tasks that handle HTML forms or "ultra-thin" clients. Only one message can be shown in response to a single event. Executing $showmessage() more than once in response to the same event will result in a single Ok message with the text and title of the last call to $showmessage being shown. Alternatively, you can use the OK message command provided it is executed on the client; in this case the command uses a standard browser alert() or confirm() dialog.

Adding Objects to JavaScript Forms

You can add a new object to a remote form instance or a Paged Pane in the form at runtime using the $add() method. Note that you cannot create an entirely new object using this method, rather the $add() method in this context lets you copy an existing object in the form and add it to the form or pane. The following method can be used, where $cinst is the remote form instance:

```
$cinst.$objs.$add(cName,rSrcItem[,rParentPagedPane,iPageNumber,bAllPanes=kFalse])
```

adds new object cName to the JavaScript remote form instance by copying rSrcItem (existing object in same instance). The default action is that the new object is added to the form (when rParentPagedPane etc are omitted), otherwise you can specify the Paged Pane parameters to add the new object to a Paged Pane in the remote form.

This method can only be used in server methods, not a client-side method, that is, the information about methods etc is not present on the client. There is a logical limit of 16384 controls on a remote form, although performance will be impaired well before that limit.

The rSrcItem and rParentPagedPane parameters must both be item references to objects in the same remote form instance: their original properties and methods defined in the class when the form was instantiated will be the initial properties and methods of the new object.

If the object to be copied (rSrcItem) is a paged pane, then its children are *not* copied.

This method of copying objects cannot be used to copy complex grids. Furthermore, complex grids cannot be anywhere in the parent hierarchy.

Note that you cannot use the $remove() method to remove objects you have added using the $add() method: to remove or hide such an object, you can set $visible for the object to kFalse. In addition, you should note that $order is not assignable at runtime so you cannot add a new object and then change its field order.

Remote Form Instance Group

The item group $root.$iremoteforms is a global group of all remote forms instantiated in Omnis at any one time. You can inspect the $iremoteforms group within the context of the current remote task (the current global group remains unchanged).

Therefore you can use $ctask.$iremoteforms in a remote task to return an item group containing both top-level remote form instances and subform instances. In addition, the $obj notation has been implemented for subform objects, so that if the current item is an item reference to a remote form instance contained by a subform object, item.$obj is the item reference to the remote form subform object. For example, if RF1 is a remote form containing a subform object named sfobj, with a classname of RFSUB, then after both forms have constructed:

```
$ctask.$iremoteforms     ## will contain instances RF1 and RFSUB
$ctask.$iremoteforms.RFSUB.$obj     ## will be $iremoteforms.RF1.$objs.sfobj
```

Remote Form Events

JavaScript fields and controls trigger events which you can respond to in the $event() method for individual controls. JavaScript remote forms also trigger events, including when the screen size or orientation of the client device changes, or when a form or subform is brought to the top when multiple forms are open. Remote forms trigger the following events:

- **evAnimationsComplete**
  The animation has completed, with the parameter **pEventCode;** the Animations section
- **evFormToTop**
  The remote form is about to become visible on the client, with parameters **pEventCode** and **pScreenSize** which is a kSSZ… constant for the current screen size on the client; see the Multiple Forms section
- **evLayoutChanged**
  (applies to kLayoutTypeResponsive remote forms) generated when the responsive layout breakpoint changes, that is, when a mobile device is rotated, or when a browser window is resized: this event is also triggered when the form first opens. This has the event parameter **pBreakpoint**, which is the integer value of the initial or new layout breakpoint (e.g. 320 or 768, the default values).
  (Note that in versions prior to Studio 8.1.6, the pBreakpoint parameter was reported as a string in a client executed method, but this is now reported as an integer value, which matches the behaviour of evLayoutChanged in server methods.)

- **evScreenOrientationChanged**
  (applies to kLayoutTypeResponsive, kLayoutTypeScreen, and kLayoutTypeSingle type forms, since Studio 10.0) The orientation of the screen has switched between portrait and landscape, with parameters **pEventCode, pScreenSize** (a kSSZ… constant for the current fixed screen size on the client, and **pOrientation** (kOrientPortrait or kOrientLandscape depending on the *resulting* orientation of the form).
- **evSubFormToTop**
  An existing remote form, contained in a subform that has $multipleclasses set to kTrue, is about to become visible on the client, with the parameter **pEventCode;** see the Multiple Forms section
- **evOpenContextMenu** and **evExecuteContextMenu**
  JavaScript remote forms report the context menu events: see the Context Menus section

Event Parameters

When an event is triggered, a number of event parameters are sent from the client to the event handling method. The first of these parameters is always the name of the event that occurred, and all subsequent parameters are specific to the event and describe the event in more detail. For example, a click on a list passes the click event in the first parameter (pEventCode=evClick) and the list line clicked in the second parameter (pLineNumber).

Enabling Form Events

If you want to use any remote form events in your code, you have to enable the events in the $events property of the remote form. You have to do this in design mode by clicking on the background of the form, selecting the Properties option, and selecting the $events property in the Property Manager. You can enable an event by selecting it in the dropdown list for the $events property.

Form Orientation

When the orientation of a remote form changes (e.g. when the end user rotates their mobile device, or in some cases resizes the browser window), Omnis sends an **evScreenOrientationChanged** event to the top remote form. This allows the remote form to adjust the coordinates of any dynamically added objects. In addition, evFormToTop also receives the pScreenSize event parameter, allowing other forms to make adjustments if necessary when they come to the top. In addition, remote forms of type kLayoutTypeScreen (non-responsive) trigger the evLayoutChanged event when their layout changes.

Event Methods

You can trap and respond to events generated in a remote form in the **$event()** method in the form. You have to add a method called $event to the Class methods for the form to create an event handler. Like other event methods you can use the *On event* command to trap specific remote form events. The following $event method responds to the evScreenOrientationChanged event and sets the iScreensize variable to the correct screen size.

```
On evScreenOrientationChanged
  Switch pScreenSize
    Case 3
      Calculate iScreensize as kSSZjs320x480Portrait
    Case 4
      Calculate iScreensize as kSSZjs320x480Landscape
    Case 9
      Calculate iScreensize as kSSZjs768x1024Portrait
    Case 10
      Calculate iScreensize as kSSZjs768x1024Landscape
    Default
      Calculate iScreensize as kSSZDesktop
  End Switch
  Do method setupSizes
```

Running Event Methods on the Client

Event handling methods can be set to run on the client and for most simple methods and calculations this is advisable; in most cases, when you add a $event method it is set to execute on the client automatically (and marked in pink, the default color). To set a method to run on the client you need to Right-click on the method in the method editor and select the 'Execute on Client' option. By default, an event is sent back to the Omnis App Server, the client is momentarily suspended while the event handling method is processed on the server, and when the method is finished control is passed back to the client. In some cases, this may not be a problem, or when server data is required, but in general it makes sense to execute your methods on the client and avoid any network delay if possible. Whether or not you execute a method on the client will depend what the method has to do and what information it requires: in general, any method that changes the user interface on the client can be executed on the client, while a method that needs to fetch data or write data to your server database needs to execute on the Omnis App Server.

Testing JavaScript Remote forms

You can test a JavaScript remote form by clicking on the **Test** button in the Design bar, or using the **Test Form** option, available by Right-clicking on the background of the form, or by pressing **Ctrl-/Cmnd-T** while the remote form is the top design window. You can also Right-click on a Remote form in the Studio Browser and select **Test Form.** (Your library must contain a Remote task and its name must be assigned to the $designtaskname property of the remote form for it to be opened: see Remote Tasks for more information about remote tasks.)

The **Test** button or **Test Form** option will open the remote form in a web browser specified as the default browser on your development computer; the remote form will open in a new browser window or create a new tab if your browser is already open. Omnis has a built-in HTTP server to allow you to test remote forms locally in a web browser.

The **Select Browser And Test Form…** option (Shift+Ctrl/Cmnd+T) on the Remote form design Context menu opens a dialog containing a list of web browsers installed on your system, including an entry for the **System Default,** allowing you to select a browser in which to test your remote form. This can be useful if you want to test a remote form in several different browsers while designing the form and testing your app, for example, to check that some JavaScript code behaves the same in all browsers. Note that the option is only present in the Context menu when your system has more than one registered web browser, otherwise the option is hidden, and the default system browser will be used via the standard **Test Form** option.

Having tested a remote form in your web browser, you can switch back to Omnis and continue to change your remote form or its methods, and use **Ctrl/Cmnd-T** at any time to test your form. Each time you press Ctrl-T Omnis will try to open a new browser window or tab. Alternatively, if your web browser is already displaying your remote form, and you have modified the form, you can switch to your web browser and **Refresh/Reload** the browser to see the latest changes to your form (but if you change JS theme you have to re-open the browser window/tab using Ctrl-T).

Default Web Browser

When you test your remote form using the **Test** button or **Test Form** option (Ctrl/Cmnd-T), it is opened in the default web browser on your development computer. If you want to test the form in another browser on your computer, you can use the **Select Browser And Test Form…** option (Shift+Ctrl/Cmnd+T) on the Remote form design Context menu; alternatively, you can copy the test URL and paste it into another browser (note the port number may change from session to session).

If you want to override the default action for the Test Form option, you can specify the name and path of an alternative browser in the $webbrowser Omnis preference (edit the Omnis preferences via **Tools>>Options** on Windows, or **Omnis>>Preferences** on macOS). If this preference is empty, then the default browser on your development computer will be used for testing remote forms.

Test Web Page

When you test your remote form using **Test Form** (Ctrl/Cmnd-T) an **HTML page** is created for you automatically containing the JavaScript Client and all the required parameters to allow you to open your Omnis app in a web browser. The test HTML file is located in the HTML folder under the main Omnis folder, and can be used or incorporated into the other web pages on your website when you are ready to deploy your application. The Web Preview displaying your form in the remote form editor also creates an HTML file in the 'html/design' folder, in order to render the page in design mode, but this is not required for deployment.

The name of the test HTML file will be *the same as your remote form class name* plus the .htm extension. The test HTML is based on a template file which is also located in the HTML folder: see below for more information about the template.

The URL for the test HTML page will be something like the following:

http://127.0.0.1:51452/jschtml/<remoteformname>.htm

The test URL contains the IP address of your Localhost (127.0.0.1), the port number of your copy of Omnis Studio, a reference to the test JavaScript Client HTML folder, and the name of the HTML file. The port number during testing will be the port number specified in the $serverport Omnis preference, or if this is empty (the default) a port number is selected randomly from the available ports on your computer.

If you try to open or navigate to the test URL from your browser history it may not work: in this case such a URL may not have the correct port setting since the port number is assigned dynamically during testing if the $serverport property is empty and therefore may be different from one session to another. In addition, you cannot open or test your remote form by opening the test HTML in the template folder: again your browser will not have the correct URL to load the test HTML file.

Omnis Studio and your library *must be open and running* to test your remote form. So if you open the test HTML file from your file system, and Omnis and your library are not open, then your remote form will not be displayed and your web or mobile app will not run.

Template HTML File

The test HTML created when you use the **Test Form** or **Ctrl-T** option is based on a template file called 'jsctempl.htm', which is located in the HTML folder under the main Omnis folder. When you press Ctrl-T a copy of the template file is made and the individual parameters for your remote form are written to the test HTML file: this occurs *every time you test your form* to ensure the test HTML file is up to date and has the correct parameters. Therefore, if you make any changes to the HTML file in HTML folder your changes will be overwritten *the next time you test your remote form*: if you want to keep a version of this file, either rename it or copy the file to another location.

See Editing Your HTML Pages for more information about the contents of the test HTML page and what changes you may need to make for deployment.

Using an Alternative Template file

The remote task class property $htmltemplate allows you to specify a different HTML template to use to test a remote form, rather than using the default template 'jsctempl.htm'. For example, you may want to create a template with your own set of parameters in the "omnisobject" <div>, but retain the default template.

The new $htmltemplate property specifies the name of a template file (which must exist in the html folder) to use when testing any remote forms that use this remote task as its design task. If $htmltemplate is empty (the default), Omnis uses the default template 'jsctempl.htm' located in the html folder, which matches the behavior in previous versions.

Debugging Remote Forms

When you open a remote form in your development browser you will need to debug the methods and code in the form. You can do this by setting breakpoints in your code and you can send messages to the Omnis trace log or the JavaScript console (provided it is available) to allow you to debug your code.

Breakpoints

You can set breakpoints in your code, so when you test your remote form using the Test Form option or Ctrl-T and a breakpoint is encountered, control will pass from your web browser back to Omnis. In this case, when a breakpoint is encountered, the Omnis entry (button) in the Windows Task bar will flash (the default color is orange) and you will have to click on the button to return to the Omnis application window to continue debugging.

Trace Log

The *tracelog()* function allows you send debugging and other messages to the Omnis trace log from within client methods executed in the JavaScript Client: this will allow you to debug client methods. The tracelog(*string*) function writes the *string* to the Omnis trace log, or does nothing if debugging is disabled using the library property $nodebug. It returns true if the string was successfully written to the trace log.

Alternatively, in JavaScript client-executed methods you can use the *Send to trace log* command which sends the text to the JavaScript console.

Runtime & Server Logging

The Library preference $alwayslog ($clib.$prefs.$alwayslog, defaults is kFalse) allows you to log messages in the Runtime and Server versions of Omnis to help you debug your code. When kTrue, the *Send to trace log* command and *tracelog()* function always write non-diagnostic messages to the trace log (overriding the check for debuggable code).

Client Caching

There is an entry in the Omnis configuration file (config.json) that allows you to control whether HTML pages are cached or not by the built-in HTTP server in Omnis (which is used for testing forms in design mode). The "preventclientcaching" item under the 'omnishttpserver' entry in the config.json file is set to true by default and prevents web pages from caching. When set to true, this would mean that every time a page is accessed, the page and any linked scripts (JS files, CSS files) are loaded or refreshed and not cached: note this is for testing purposes only, and does not apply when you deploy your app. If you want pages to be cached you can set this item to false.

The "preventclientcaching" entry in config.json has the following format:

```
"omnishttpserver": {
        "preventclientcaching": true
    }
```

When hosting your files on a web server (as recommended for deployment), this setting does not apply - your web server will have its own settings to control client caching behavior of files it serves.

Testing your Remote Form on a Mobile Device

To test your remote form on a mobile device or any other client apart from your development computer, assuming those devices are within the same local network (LAN/WLAN) as your development computer, you can enter the test URL into the web browser on your device but replace the Localhost IP address (127.0.0.1) with the IP address of your development computer. For example, reusing the test URL above and replacing the IP address, the following URL could be used on a mobile device such as a phone or tablet computer:

```
http://194.131.70.184:51452/jschtml/jsMain.htm
```

You can use the ipconfig command to find the IP address of your development computer, via the Command prompt on a PC or the Terminal on a Mac.

You can test a mobile remote form in a wrapper application using the *Test Form Mobile* (Ctrl-M) option, assuming a wrapper application is setup and enabled: if a wrapper is not setup you can test your mobile forms in a web browser during development, as above. See the Deployment chapter for details about setting up a wrapper application.

Client Script Version Reporting

If the build version of the scripts in the JavaScript Client is different to the scripts on the server, then the mismatch is reported as an error.

If there is a **major version** difference between client and server, an error message is generated, and the client will not run in this situation. The error message text is determined by a new localizable string "omn_cli_script_majorversion_mismatch" in strings_base.js.

If there is a difference in the **build revision**, a warning is logged to the browser console describing the issue. For example, if you patch the scripts only (rather than installing a new version of Omnis Studio), then this difference will be logged in the console.

Troubleshooting Remote Forms and Tasks

Remote Tasks

When I try to test my remote form using Ctrl-T, it does not open and the error "To use a remote form class, you must set the design task of the remote form class to a remote task" is displayed.
You need to create a remote task in your library and set the $designtaskname property of the remote form class to the name of the remote task you created. A remote form instance needs a remote task instance to run (for testing or deployment), so will not open without a remote task and the $designtaskname property being set.

Remote Forms

When I open my blank remote form in design mode I don't see any JavaScript components in the Component Store.
All new remote forms should be set to run on the JavaScript client, but if for some reason the $client property is not assigned, you need to set it to kClientJavaScript to use the new JavaScript components. (Note you cannot switch an existing Web Client based remote form to the JavaScript Client, but there is a migration tool available in the Studio Browser to help you move to the new client.)

Events

I have placed an event method behind my button (or any other event-driven object), but nothing happens when I click it while testing the form in a browser.
You must specify which events are to be triggered by the object or remote form in the $events property of the object or form. So for a button, the evClick event must be checked in the $events property of the button. Events for some objects are checked by default, but you may like to examine the state of each event in the $events property and make sure the events you need are enabled.

Client Methods

You can run certain methods on the client, such as event handling methods ($event), instead of running them on the Omnis App Server: these are referred to as *Client-Executed Methods* or simply **Client Methods.** When such client methods are called in your application, runtime execution *does not pass back to the Omnis App Server*, rather the method is executed *entirely in the end-user's browser* within the JavaScript Client. Enabling certain methods to execute on the client can speed up your application, by cutting down on network traffic, while from a design point of view, using client methods allows you to add more interactivity into the UI of your web or mobile app, including using native JavaScript in your Omnis code.

Any methods in your JavaScript remote forms that are enabled to execute on the client are converted to JavaScript files. These JavaScript files are run in the browser when the method is called on the client. When a client browser opens a JavaScript remote form, the Omnis App Server generates a JavaScript file containing the client methods for the remote form (the generation of this file only occurs once unless the remote form class is modified, in which case it will regenerate the file). This file is added to the html/formscripts folder in the Omnis tree inside a subfolder named after the library. Each remote form containing client methods has a separate JavaScript script file.

When you deploy and run your web or mobile app on the Omnis App Server, these JavaScript script files are generated at the same location as in the development version, the first time the client calls the client-side method. For the deployed app, the JavaScript files are in minified form, and therefore are smaller and will run faster since all the comments are stripped out.

There is a sample app called **Client Methods** in the JavaScript Component Gallery, and under the **Samples** option in the **Hub** in the Studio Browser.

Enabling Client methods

To enable a method to execute on the client, you can Right-click on the method name in the Method Editor and select the 'Execute on Client' option. Whether or not an existing method can be enabled to execute on the client will depend on the Omnis commands, functions, and in some cases the type and scope of the variables used in the method. When you try to enable the method to execute on the client, Omnis checks whether all the commands and functions within the method can be executed on the client, and if not, it will not allow the method to be marked as 'Execute on Client'. If the method can be executed on the client, the method name will be shown in pink (the default color) and the restrictions on client methods will then apply to this method.

If you create a new empty method in the method editor, it can be enabled to execute on the client without error (since it does not contain any commands, functions or variables at this stage), but the number of commands and functions available to use will be limited to those shown in the Method Editor. The type of Local and Parameter variables you can create will also be restricted: see *Data Types* below.

The Omnis Help (F1) indicates whether or not a command or function can be executed on the JavaScript client.

Binary functions

The Binary functions cannot be used in JavaScript client methods since JavaScript does not support binary data particularly well.

Text Blocks

You can use the text block commands (Begin text block, Text:, End text block, Get text block, JavaScript:) in client methods to build up a block of text, e.g. to generate styled text (see the Styled Text section).

Object Properties in Client Methods

While some properties can only be assigned in Server methods, you can read the current value of many more remote form properties and component properties in client-executed methods running in a form instance. So for example, you cannot change the size and position of an object in a client method (by setting $top, $left, $width, and $height), but you can return the current values for an object in a client method, e.g. $cobj.$width returns the width of the object.

An error will be generated if you try to read the value of a remote form/control instance property from a server-executed method: “Cannot get the value of a remote form instance property when executing code on the server”.

Code Block Error Checking

JavaScript code generation for client methods detects missing block terminators, and if an error is found, Omnis adds an error to the Find and Replace log and opens the log. For example, an error is generated if there is a While loop with no End While, or an If with no End If. You can open the method containing the error by double-clicking the error message in the Find and Replace log.

Auto Client Executed

When you add a new method in the method editor, and the method editor recognizes the method name as a known method that should be client-executed, then the method editor marks it as client-executed, and also adds any parameters (if required). This applies to new methods with the following names: $candrop, $drag, $filereadcomplete, $filtergrid, $getscrolltip, $init, $pushed, $sfscanclose, $sfsorder, $sortgrid, $sqldone, $term. In addition, the $sqldone method receives some boilerplate or template code when you add it.

$init method

You can create a client-side method with the name $init in your remote form which the client calls after the form and the client scripts file have been loaded. This allows you to do any final initialisation of the remote form, especially when the remote form is running in serverless client mode. When you create or enable the $init method it will be marked as client executed automatically.

Leave Site Prompt

You can use the $init() client method to add a ‘Leave Site’ prompt to your application which forces the browser’s built in prompt to be shown when the end user attempts to close the remote form if there has been any sort of interaction. To add a ‘Leave Site’ prompt, add a *Quit Method* command returning any string to the $init method of your remote form, for example, Quit method ‘Show leave site prompt’. The default ‘leave site’ text for the browser is displayed, so the text you add to Quit method is irrelevant.

Event Specific Client Methods

You can use the **$eventclient()** method to call a client-executed method *for specific named events*. The client will run the $eventclient method if it contains *On default* or an *On <event>* statement for the named <event> that has been triggered, otherwise it will run the $event method, which must be server-executed if $eventclient is specified.

When specifying $eventclient, as with $event, events are still only sent when they are present or enabled in the **$events** property for the control or form. Omnis calculates the subset of $events to be handled by $eventclient as those events specified in an On statement or On default statement in $eventclient, at any level in the inheritance hierarchy for the control or form. In this case, if you use On default, all events are processed by $eventclient.

Any events not present in the subset of $events to be handled by $eventclient are sent to $event for the control or form and run on the server, unless they are one of the small set of client-only events, e.g. $candrop and $drag.

The current event processing model, where all events go to $event, which is either client or server executed, works as in previous versions if there is no $eventclient method present for the control or form.

When using inheritance, you can also override $eventclient in a subclass.

Showing Built-in Methods

When the **Show Built-in Methods** option is enabled (the default), the method editor tree list shows all possible *built-in class and control methods* that can be overridden, including $event and $eventclient (in versions prior to Studio 11 most of these methods were available but not listed). Note that control methods that are built-in to JavaScript client objects cannot be overridden, meaning they are not displayed in the method tree list when showing built-in methods. In addition, built-in methods in the tree have a tooltip that is displayed when the **Show Method Content Tips** option in the View menu is enabled.

Debugging Client methods

Client methods cannot be debugged in the Omnis method editor since method execution occurs in the client browser. Instead, you must use the Script Debugger within the browser you are using to test the client executed parts of your application. Note that for this reason you cannot use any Omnis commands that interact with the debugger or trace log (Trace on, Open trace log, etc) in client methods, or any that would interrupt command execution (Breakpoint, Yes/No message). However, while developing your app, you can use the *Send to trace log* command in client executed methods to write a line to the JavaScript console.

Client methods cannot be debugged in the Omnis method editor since method execution occurs in the client browser. Instead, you must use the Script Debugger within the browser you are using to test the client executed parts of your application. Note that for this reason you cannot use many Omnis commands that interact with the debugger or trace log (*Trace on, Open trace log,* etc) in client methods. However, while developing your app, you can use the *Send to trace log* command in client executed methods to write a line to the JavaScript console.

The *Breakpoint* command can be used in client-executed methods to set a ‘hard’ breakpoint in the code, but note that this will only be hit if the web browser developer tools are open. It will then break into the browser’s debugger, in the JavaScript code which was generated from your client-executed method. The browser dev tools can usually be opened using the F12 key.

Errors

Omnis maintains two global system variables #ERRCODE and #ERRTEXT that report error conditions and warnings to your methods. Fatal errors set #ERRCODE to a positive number greater than 100,000, whereas warnings set it to a positive number less than 100,000. You can return the values of #ERRCODE and #ERRTEXT in client executed methods in the JavaScript using the functions errcode() and errtext(), so it is possible to write methods to handle errors when they occur.

Calling Custom Methods

You can use the *Do method* command to call a custom method (your own method) in the current remote form instance. If you use this command in a client method you must include parenthesis after the method name for the method to be called. For example, to call a custom method called $mymethod from within in a client method in the current remote form instance, you must use *Do method $cinst.$mymethod(),* or you can use *Do $cinst.$mymethod().*

Data Types for Client methods

JavaScript itself only supports a small number of built-in data types, and these do not correspond directly with Omnis data types. Since instance variables are used by both the JavaScript Client and the Omnis App Server, these can still be given any Omnis data type.

However, *Local* variables and *Parameter* variables in client methods can only be Date Time, List, Row or Var type. The Var type is a generic variable used for data of any type. The client does not enforce the type of data stored in a variable, unless you assign a value to an instance variable or list column, in which case the client will convert the assigned value to the correct Omnis type for the instance variable or list column, or failing that, if the data cannot be converted, the client throws an exception, reported by an OK message.

Further restrictions

- Field reference parameters are not supported, since JavaScript methods use call by value.
- Date Time local variables and parameters do not have a subtype in client methods.
- Date Time, List and Row variables are represented by a JavaScript object on the client, whereas the other data types are the native JavaScript types (number, string and Boolean).

When you change a method from client to server execution, or back again, the data type of Local variables and Parameters changes to the most logical supported value for client or server execution (restoring the original type when changing from client to server execution if the type on the client is Var).

Variable References in Client Methods

There are some subtle differences to the way variables are passed around in client methods in Omnis Studio, as opposed to methods that are executed on the server. This is due to the fact that client-executed methods are running JavaScript code, and so are bound by the rules of that language.

- In JavaScript, variables and method parameters are always passed by value. This means that a *copy of the value* of the variable is passed around.

This is easily understood for **primitive** data types, such as string, number, and boolean, but **objects** are slightly more nuanced, that is, anything that's not a primitive, including Omnis Lists, Rows, Dates and Remote Objects.

Object-type variables

- The *value* of an object variable is essentially a *pointer* to that object.

This means that it is cheap and efficient to pass objects around, but there are two important consequences:

1. If you change some property of a passed object, it will change the property on the original object too.
2. If you re-assign the **whole variable** of the passed object, you are replacing the pointer, so will not effect the original object.

In Omnis, this only comes into play when you are using local variables or parameters in a client-executed method.

Example 1

```
Calculate lRow1 as iRow
Calculate lRow2 as lRow1
# lRow1 & lRow2 are two separate local variables, but because they are object types,
# they are now pointing to the same object in memory (as one was assigned the value of the other).
Calculate lRow1.C1 as "TEST"
# Changing an aspect of lRow1 will now also effect lRow2. So setting column 1 of lRow1 to "TEST"
# also sets column 1 of lRow2.
Calculate lColValue as lRow2.C1
# lColValue is "TEST"
```

Example 2

```
Calculate lRow1 as iRow
Calculate lRow2 as lRow1
# lRow1 & lRow2 are two separate local variables, but because they are object types
, # they are now pointing to the same object in memory (as one was assigned the value of the other).
Calculate lRow1 as row("TEST")
# Because we have reassigned the whole lRow1 variable, it has created a new pointer
# as the value of lRow1, so is no longer pointing to the same object as lRow2.
Calculate lColValue as lRow2.C1
# lColValue is the original value of column 1 from iRow
```

Execution and Method Calls

Method execution in the JavaScript client means that that no user-visible updates to the user interface occur until control returns from the executing JavaScript to the browser (or other container). This means that client methods should be as short as possible. This also means that if a client-side method calls a server method, this call cannot occur in a synchronous manner, since the user interface would appear frozen while the server code was executing (since no user-visible updates to the user interface occur while a synchronous AJAX call is running). This means that the mechanism for calling a server method from a client method is implemented differently for the JavaScript client.

When code in a client-side method calls a server method (either using $cinst.$methodname notation, or by using the *Do method* command), the call to the server is executed *asynchronously*. While the server method is executing, the UI is made inaccessible using a loading overlay – the exception to this is that $alwaysenabledobject and edit controls calling a server-executed evKeypress are left accessible. After making the server method call, the client-side method continues to completion (note that the server method call returns true or false to indicate if it has started). The client-side method can only call one server method like this – if it makes a second call, the client throws an exception.

Return Methods

If the client code needs to handle the return value from the server method, then you must implement a client-side method in the remote form with the name <method name>_return. For example, to receive the return value from a server method called $test, implement a client-side method $test_return; the server method ($<method name>) must be called from a client-side method. The <method name>_return method must have a single parameter, which is the return value from the server method. A typical action of the _return method might be to update a progress control, and then issue another call to the server. This allows a time-consuming operation to execute while showing its progress.

The **Insert Client Return Method After** option in the **Modify>>Method** menu (and on the Method list context menu) allows you to add a "<method name>_return" method for a server method; the method is named <method name>_return automatically and placed after the server method.

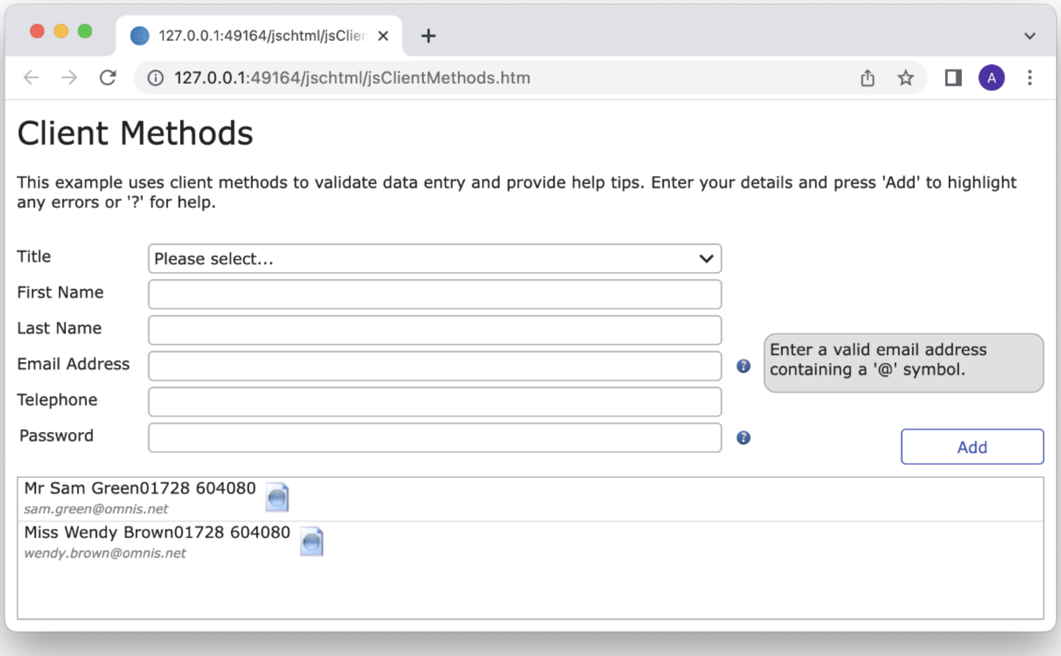
From Studio 10.2 onwards, you can create such return methods for controls (fields). If a client-executed method calls a server-executed method on a control, when execution returns to the client, Omnis will look for the <method name>_return method *on the control,* and if not found it will look for the return method at the form level, as in previous versions.

Object and Library names

You can use $cinst.$class or $cinst.$lib in client-executed methods to get the name of the current class or library, where $cinst is a JavaScript remote form instance executing the method.

Client Methods Example

There is an example app demonstrating **JS Client Methods** in the **Samples** section in the **Hub,** and the same app is available in the **Omnis Components** app in the Apps Gallery on the Omnis website (www.omnis.net/platform/#jsgallery)**.** The app includes a simple entry form with client-side field validation, and some animated help tips.


127.0.0.1:49164/jschtml/jsClier
127.0.0.1:49164/jschtml/jsClientMethods.htm
Title
Please select...
First Name
Last Name
Email Address
Telephone
Password
Add

Client Methods

This example uses client methods to validate data entry and provide help tips. Enter your details and press 'Add' to highlight any errors or '?' for help.

Enter a valid email address containing a '@' symbol.

Mr Sam Green01728 604080
sam.green@omnis.net

Miss Wendy Brown01728 604080
wendy.brown@omnis.net

Figure 92:

You should open the library in the Hub or download the library and examine the client methods and code. A sample of the code is described here.

Field Validation

A web or mobile form may include several mandatory fields. You can use a client-side method behind the Submit button (called Add in the example app) to check that the fields contain some data – this is the code for the email field:

```
# $event method behind Submit button - it is enabled to execute on the client
On evClick
  If pos('@',iEmail)=0
    Do $cinst.$objs.EmailLabel.$textcolor.$assign(kRed)
    Calculate lCurField as 'Email'
  Else
    Do $cinst.$objs.EmailLabel.$textcolor.$assign(kBlack)
  End If
  Do $cinst.$setcurfield(lCurField)
```

The code checks whether or not the Email field contains an '@' and if not the field label is colored red and the cursor is placed in the Email field – if the Email field contains data the method moves onto the next field. All the other fields on the form are checked for any content and handled in a similar way: the password field is checked for a minimum number of characters, as follows:

```
If len(iPassword)<6
  Do $cinst.$objs.PasswordLabel.$textcolor.$assign(kRed)
  Calculate lCurField as 'Password'
Else
  Do $cinst.$objs.PasswordLabel.$textcolor.$assign(kBlack)
End If
Do $cinst.$setcurfield(lCurField)
```

Animated Help tips

In the **JS Client Method** example app, the form has some help buttons that the end user can click on to receive tips about how to fill out the form. These help tips are implemented using a hidden text box and the animation methods – when you click on a button, the text box is made visible and is moved into position next to the appropriate field, and when the button is clicked again the text box is hidden. The code to achieve this is in a client-side method behind the Help tip button marked with a '?' icon: the button is in fact a Picture control with its $iconid set to 1794, and the code is placed behind the $event() method. The Tip box is a paged pane containing a Rounded rectangle and a Text field – when the form is opened its $alpha is set to 0 and it is positioned off the screen. The help tip next to the Password field has the following event method:

```
On evClick
  If iLastTip='password'
    Do $cinst.$beginanimations(500,kJSAnimationCurveLinear)
    Do $cinst.$objs.Tip.$alpha.$assign(0)
    Do $cinst.$commitanimations()
    Calculate iLastTip as ''
  Else
    Calculate iTip as "A password must to at least 6 characters long."
    Do $cinst.$beginanimations(500,kJSAnimationCurveLinear)
    Do $cinst.$objs.Tip.$top.$assign(270)
    Do $cinst.$objs.Tip.$alpha.$assign(255)
    Do $cinst.$commitanimations()
    ; the Tip box is made visible and moved into position
    Calculate iLastTip as 'password'
  End If
```

There is a similar Help tip button and method for the Email field. If the Password Tip box is already visible in the form and you click the Email Help button, the Tip text is changed and the Tip box itself is moved next to the Email field.

Client Objects and Inheritance

The JavaScript client uses JavaScript objects to represent each remote form and each control on the remote form. The client methods of a remote form and its controls are defined as methods of the remote form and control objects.

Once a remote form and its client methods script file have been loaded, the JavaScript remote form object has a member called ivars which is a JavaScript object that maps instance variable names to instance variable field numbers. If you override an instance variable, then the ivars object will have more than one entry for the variable – overridden superclass variables will have a numeric suffix to distinguish them.

The *Do inherited* command is evaluated at the point at which the script file is generated by the server. This means you can only use *Do inherited* from within the actual method you have overridden. This differs from the other clients, in that they allow you do call *Do inherited* from a private method called from an inherited public method, and then the call to *Do inherited* calls the overridden public method.

New Page Browser Prompt

In most desktop browsers (excluding Opera) a web page can prompt the user before navigating to another page. Using the $init client-side method for a remote form you can specify the message to be added to the user prompt.

If the $init method returns a character string, the browser will prompt before navigating to another page, *but only if the browser supports this feature*. The returned character string may not be in the browser prompt, depending on the browser, for example, Firefox does not currently display it.

If $init does not end with a *Quit method* command, or returns another type, then functionality is unchanged from the current behavior.

Once one $init method has returned a string, any return values from any other $init calls for other remote form instances are ignored, so there is no way to turn off the prompt once it has been enabled.

Client Commands

The **$clientcommand()** remote form method allows you to execute various pre-defined commands on the client device, including ones that open various types of message boxes, and ones that can access functions on a mobile device. The suitability of certain client commands will depend on the current client device (e.g. some of the commands may not be available for certain mobile devices), and due to the differences between web browsers on desktop computers and phones, so *these client commands should be thoroughly tested,* as appropriate, for all platforms or devices you wish to support.

These client commands should be executed in the context of the current *remote form instance* using **$cinst,** and require various parameters that depend on the command sent to the client. The client commands *must be executed on the client;* they have the general syntax:

```
Do $cinst.$clientcommand(cCommand,wRow)
```

where $cinst is the current remote form instance, *cCommand* is the name of the client command to be executed on the client, and *wRow* is a row variable containing one or more parameters to be passed to the client.

The following is a summary of all the client commands (in alphabetical order), and they are described after the table (in functional groups):

Client command	Description
assignpdf	Assigns the specified PDF to the specified HTML control
clearerrors	Clears all errors set with $errortext for all objects on the form
clearlocale	Clears the locale on the client
closefile	Closes a file or files supplied by the drag value of kDragFiles to evDrop
closepush	Closes an open push connection to the Omnis Server

Client command	Description
enablepushnotifications	Enables or disables push notifications on the client inside a JavaScript mobile wrapper; see the wrapper docs on the download page
javamessage	Shows a message box on the client
loadpreference	Loads a named preference value from the client preferences into an instance variable
lockui	Locks the client user interface
noyesmessage	Opens a no-yes message box
okcancelmessage	Opens an ok-cancel message box
openpush	Establishes a push connection to the Omnis Server for the remote task
playsound	Plays a sound on the client (unlikely to work on mobile devices)
readfile	Reads and then closes the file identified by file ident (supplied to evDrop)
savepreference	Saves a value (as a character string) as a named preference on the client
setcustomformat	Sets the default custom date format used when $dateformatcustom is empty
setlocale	Sets the locale on the client
settheme	Sets the theme for the JS client, i.e. all remote forms in the library or application
showloadingoverlay	Shows or hides a loading overlay over a control or the entire form
showpdf	Opens the specified PDF in a new browser window or tab
showtoast	Activates a “toast message” which displays a message to the user in a small popup which disappears after a timeout, either 5000ms or specified amount
soundbell	Sounds the bell (unlikely to work on mobile devices)
speakmessage	Speaks a message, useful for visually impaired users
subformdialogclose	Closes the topmost subform dialog
subformdialogshow	Opens a single subform as a modal dialog
subformpaletteclose	Closes a subform palette
subformpaletteshow	Opens a subform palette
subformset_action	Where action is one of the following: add, formadd, formremove, formtofront, or remove, e.g. subformset_add adds a set of subforms to the form. See *Subform Client Commands*
yesnomessage	Opens a yes-no message box

Message Dialogs

There are a range of message dialogs available including dialogs for **Yes/No** messages (Yes is default button), **No/Yes** messages (No is the default button), and **Ok/Cancel** messages (OK is the default, with an optional Cancel button).

There is an example app called **Dialogs** in the **Samples** section in the **Hub** in the Studio Browser that shows how to use the client commands to open message dialogs, plus the same library is available in the JavaScript Component Gallery. In addition, the speakmessage client command can speak a message.

The message box client commands allow you to enter the *message text* in the first column of the row variable. You can create a line break in the message text using //.

Multiple calls to any of the message commands during the processing of a single event will result in *only the last call having any effect* and that message being shown, so all previous messages will be ignored or overwritten.

Yes/No Messages

The "yesnomessage" command opens a **Yes/No** message box in which **Yes** is the default button.

```
Do $cinst.$clientcommand("yesnomessage",rowVariable)
```

Where *rowVariable* is row(cMessageText, cTitleText, cPublicFormMethodNameCalledOnYes, cPublicFormMethodNameCalledOnNo [,cPublicFormMethodNameCalledOnCancel]); if the latter is omitted no Cancel button is shown.

For example, in the **Webshop** sample app a Yes/No message is generated using the $clientcommand method when a user clicks on a product size/type that is not available:

```
Do $cinst.$clientcommand("yesnomessage",row(con('Would you like to order >',iProductList.product_size_1,'< ins
```

No/Yes Messages

The "noyesmessage" command opens a **No/Yes** message box in which **No** is the default button.

```
Do $cinst.$clientcommand("noyesmessage",rowVariable)
```

Where *rowVariable* is row(cMessageText, cTitleText, cPublicFormMethodNameCalledOnYes, cPublicFormMethodNameCalledOnNo [,cPublicFormMethodNameCalledOnCancel]; if the latter is omitted no Cancel button is shown.

Ok/Cancel Messages

The "okcancelmessage" command opens an **OK/Cancel** message box in which **OK** is the default button, with an optional Cancel button.

```
Do $cinst.$clientcommand("okcancelmessage",rowVariable)
```

Where *rowVariable* is row(cMessageText, cTitleText, cPublicFormMethodNameCalledOnOk, [,cPublicFormMethodNameCalledOnCancel]; if the latter is omitted no Cancel button is shown.

Message Dialog Header Styling

The styling for dialog headers is defined in the classes: .omnis-wf-title.typeheader and .omnis-wf-title.typebody in the core.css file, which can be modified by overriding them in user.css.

speakmessage

The "speakmessage" client command tells assistive technology to announce a message, which can, for example, be used to convey information to visually impaired users.

```
Do $cinst.$clientcommand("speakmessage",rowVariable)
```

Where *rowVariable* is row(cMessage, bInterruptCurrentSpeech).

JavaScript Message Boxes

The “javamessage” command shows a JavaScript message box on the client. The message box can be various styles (error, warning, success, prompt, message, or query) and can have up to three buttons and accompanying text.

```
Do $cinst.$clientcommand("javamessage",rowVariable)
```

Where *rowVariable* is row(‘error’|‘warning’|‘success’|‘prompt’|‘message’|‘query’, cMessageText, cTitleText, bOpenAtMouse, cButt1text:servermeth cButt2text:servermethodname, cButt3text:servermethodname). For example:

```
Do $cinst.$clientcommand('javamessage',row('query','Update file?','Warning',kFalse,'No:','Yes:$call_next_metho
```

Dialog Icons

The message dialog displayed using the ‘javamessage’ command contains a standard icon from the material design set. Types ‘error’, ‘warning’, ‘success’, ‘prompt’ and ‘query’ all contain an icon specific to that type, while ‘message’ does not use an icon. The images for the icons are defined in the core.css file under the .typeicon classes and can be overridden in user.css.

Using a Promise

The javamessage, yesnomessage, and noyesmessage client commands (as well as the $showmessage method) return a JavaScript promise when the methods are executed on the client; a promise contains a value that can be used in JavaScript code in your remote form, for example, to initiate a specific action.

The promise’s resolve function is passed a parameter whose value depends on the message type being shown, as follows:

Method	Value returned
javamessage client command	The button number which was clicked (1-3)
yesnomessage client command	true if ‘Yes’ was clicked, else false
noyesmessage client command	true if ‘Yes’ was clicked, else false
$showmessage method	true

For all these dialog functions which return a promise, the calls will only return a promise when executed on the client. A promise will be ‘resolved’ when its dialog is closed. You can add code to run at this point using JavaScript, for example:

```
Do $cinst.$clientcommand("yesnomessage",row("Are you sure?","Really?!")) Returns lPromise
JavaScript:lPromise.then((lResult) => {
Do $cinst.$showmessage(con('You clicked ',lResult))
JavaScript:});
```

Playing Sounds

The system bell

The “soundbell” command plays the default system sound on the client.

```
Do $cinst.$clientcommand("soundbell",rowVariable)
```

In this case, rowVariable is empty.

Play a sound file

The “playsound” command plays a sound on the client.

```
Do $cinst.$clientcommand("playsound",rowVariable)
```

Where *rowVariable* is row(cNameOfSoundFile1[,cNameOfSoundFile2,...]); the sound file(s) should be located in the html sounds folder.

Browser support

In some cases, these sounds will not work in Safari on macOS or iOS since it generally only allows sounds that are in direct response to a user action. Therefore, sounds triggered in a server method are unlikely to work, whereas if they are triggered in a client-executed method they are more likely to work. In general, sounds tend to work in Chrome on macOS or iOS, regardless of how they are triggered.

In all cases, using the client commands to play sounds should be thoroughly tested, for all platforms and browsers you wish to support.

Date Format

The “setcustomformat” command allows you to set the date format used on the client when $dateformatcustom is empty (defaults to D m y).

```
Do $cinst.$clientcommand("setcustomformat",rowVariable)
```

Where *rowVariable* is row(cDateFormat). See the Date and Time Formatting section for more details about setting the data format on the client.

Client Preferences

The following commands allow you to save and load end-user data on the client, such as user preferences. You could use these commands to store and load usernames and/or passwords to allow the end user to log onto your application. As with other client commands, these commands must be executed on the client.

There is an example app called **JS Preferences** in the **Samples** section in the **Hub** in the Studio Browser showing how you can use these client commands to save and load user preferences in a remote form; the same app is in the JS Component Gallery.

Saving preferences

The “savepreference” command saves a value (as a character string) as a named preference on the client.

```
Do $cinst.$clientcommand("savepreference",rowVariable)
```

Where *rowVariable* is row(cPreferenceName, cPreferenceValue [,StorageType]).

Loading preferences

The “loadpreference” command loads a named preference value from the client preferences into an instance variable.

```
Do $cinst.$clientcommand("loadpreference",rowVariable)
```

Where *rowVariable* is row(cPreferenceName, cInstanceVariableName [,StorageType]), where cInstanceVariableName is a quoted string containing the name of the variable.

The following method can be used for a ‘Save’ button which saves the values in three fields on a form:

```
# lPrefRow Row var, iPref1, iPref2, iPref3 instance Char vars (on the form)
On evClick
  Do lPrefRow.$define(lPrefName,lPrefValue)
  Do lPrefRow.$assigncols('omnis_pref1',iPref1)
  Do $cinst.$clientcommand("savepreference",lPrefRow) Returns #F
  Do lPrefRow.$assigncols('omnis_pref2',iPref2)
  Do $cinst.$clientcommand("savepreference",lPrefRow) Returns #F
  Do lPrefRow.$assigncols('omnis_pref3',iPref3)
  Do $cinst.$clientcommand("savepreference",lPrefRow) Returns #F
```

Storage Type

The row variable passed to the "savepreference" and "loadpreference" client commands can have a third parameter, the "storage type", which allows *temporary, session,* or *local* storage options. This allows you to store text values in the client browser, either temporarily or persistently in the browser using JavaScript sessionStorage or localStorage. Storage type is of type character and can have the following values:

- "temp"
  temporary storage stored within an instance of this connection, will be cleared on page close or reload
- "session"
  JavaScript sessionStorage cleared when page session ends, survives page reloads and restores
- "local" (the default if no value supplied)
  JavaScript localStorage has no expiration time and survives page closures. When used with the wrappers the values will be shared between online and offline mode

For example:

```
Do lPrefRow.$define(lPrefName,lPrefValue,lStorageType)
Do lPrefRow.$assigncols('omnis_pref1',iPref1,"session")
Do $cinst.$clientcommand("savepreference",lPrefRow)
Do lPrefRow.$assigncols('omnis_pref1','iPref1',"session")
Do $cinst.$clientcommand("loadpreference",lPrefRow)
```

Locking the User Interface

The "lockui" command lets you lock the user interface manually, for example, when a series of events are taking place, and unlock the UI when the events have completed.

```
Do $cinst.$clientcommand("lockui",rowVariable)
```

Where *rowVariable* is row(bLock); kTrue to lock the UI, or kFalse or empty to unlock.

This command is useful with the progress control and $sendcarryon – in this case, you can lock the UI when you start the progress bar, and unlock it when you have finished using the following code:

```
# to lock:
Do $cinst.$clientcommand("lockui",row(kTrue))
# to unlock, either:
Do $cinst.$clientcommand("lockui",row(kFalse))
# or
Do $cinst.$clientcommand("lockui",row())
```

Custom Loading Indicator

The "showloadingoverlay" client command allows you to add a loading indicator (animated image and text) over an individual control, or the entire page in the JavaScript Client. As well as providing feedback to the user, that a long running operation may be in progress, it will also prevent user input. It is useful if you are doing any asynchronous operations, such as populating a list using a SQL worker object.

There is an example app called **Loading Overlay** in the **Samples** section in the **Hub** in the Studio Browser showing how you can use the loading overlay client command; in addition, the same app is in the JS Component Gallery.

The showloadingoverlay client command is executed using the $clientcommand method, as follows:

```
Do $cinst.$clientcommand("showloadingoverlay",rowVariable)
```

Where *rowVariable* is row(bShow, cControlNameOrEmpty, cMessageText [, cCSSClass]).

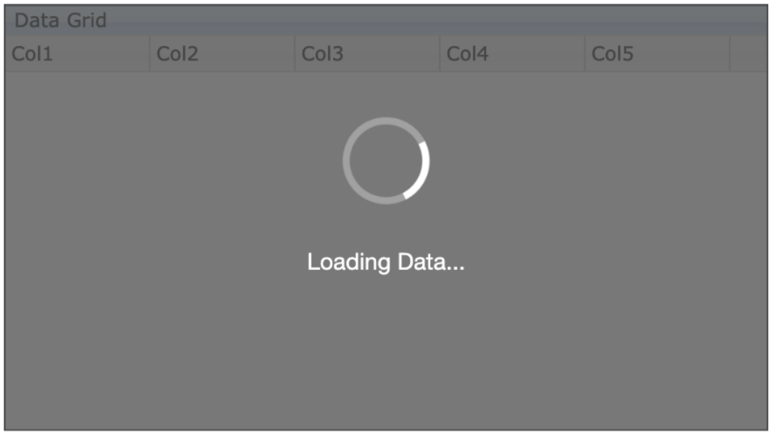


Figure 93:

- **bShow:** A Boolean value kTrue to show the overlay, or kFalse to hide it.
- **cControlNameOrEmpty:** The $name of a control on the form to which the overlay should be attached/removed from. Pass an empty string to target the entire page.
- **cMessageText:** (Optional) A string of text to display in the overlay.
- **cCSSClass:** (Optional) A CSS class to apply to the overlay. Allows you to customise the appearance of the overlay using CSS (See below).

By default, the overlay will darken whatever is behind it, and display a spinner and text string. If you wish to customize the appearance, you can do so with CSS. Create a CSS class in your **user.css** file, and pass this class name as the *cssClass* parameter.

The loading overlay comprises a toplevel div, which will be given your CSS class name. This contains a div with the CSS class name of "**container**", which contains a div with the "**indicator**" class and a 'p' element with the "**message**" class. You will need to use CSS to style all of these. Look at the 'jsLoadingOverlay' JS form in the HUB sample library which contains 'getUserCss' and 'getUserCss2' methods which build up examples of the necessary CSS and may be useful to form the basis of your CSS.

Subform Sets

There are a number of client commands, all prefixed 'subformset_', that allow you to create and manage subforms in a subform set: see the *Subform Client Commands* section for more information.

Subform Palettes

Subform palettes can be opened or closed using **subformpaletteshow** and **subformpaletteclose,** or a subform palette can be closed by clicking outside the subform: see Using Subform Palettes.

PDF Printing

The **showpdf** and **assignpdf** client commands allow you to print and display PDF files in the client browser: see the PDF Printing in the JavaScript Client section for more information.

Push Connections

Remote tasks can have a single "push connection", established using the client command **openpush,** to allow you to send data to the client: see the Push Connections section for more information.

Dragging and Dropping files

The **readfile** and **closefile** client commands enable files to be read from evDrop event while dragging files from the end user's system. See Dragging and Dropping files for more details.

Push Notifications

The **enablepushnotifications** client command enables and disables push notifications for mobile apps: this is described in a separate doc available with the JavaScript Wrapper download.

Toast Messages

The **showtoast** client command activates a “toast message” which displays a message to the user in a small popup which disappears after a timeout, either 5000ms or specified amount: see the Toast Messages section for more information.

Locale Commands

The **setlocale** and **clearlocale** client commands can be used to manage the locale on the client: see the Localizing Remote Forms section for more information.

Theme Command

The **settheme** client command can be used to set the JS Theme in the JS Client: see the Changning the Theme section for more information.

Remote Menus

The *Remote Menu* class allows you to add various types of menus to remote forms and individual controls. A remote menu can be opened as a Popup Menu control, added to a Tab Control or Split Button, or opened as a Context menu for the remote form itself or individual controls (Popup menus, Tab menus, and Split buttons).

There is an example app showing how to use a Remote menu in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

Creating Remote Menu Classes

When you design a Remote menu, you need to create a title, specified in the $title property, and add individual menu lines: the text for a menu line is added to the $text property for the menu line.

To create a Remote Menu

- Click on the New Class>>RemoteMenu option in the Studio Browser
- Click into the menu header and enter the text for the menu title
- Press Return to create a new menu line, or Right-click on the menu header and select Add Line
- After creating a new menu line, assuming it is selected, you can type the text for the menu line; alternatively, you can click into the $text property in the Property Manager and enter the text for the menu line
- To create another menu line, you can press Return or Right-click on a menu line and select **Add Line;** you can use the same context menu for delete a line
- To add a dividing line, add a new line and do not add any text; you can drag menu lines or dividing lines to reorder the options in the menu.

A remote menu does not contain methods, rather each line has a separate ID which is specified in the $commandid for the menu line. When a menu line is selected at runtime, you can use the line ID to detect which line was selected and branch your code accordingly. For example, when a remote menu is used as a popup menu and the menu is clicked, the evClick event is triggered and the value of $commandid for the selected menu line is reported in pLinenumber.

Menu Line Icons

You can add icons to Remote menu lines and can be chosen when you create the remote menu class, along with the text for the menu line (they must be 16x16 if using PNG icon images). The icon in each menu line is specified by $iconid. Checked menu lines use the checked state of the icon if the icon is multi-state.

Note that $objs.$add for a remote menu instance does not have a parameter to add an icon id. You can only set this after adding the menu line, by assigning $iconid for the new line (since the new menu line needs to reference the icon on the server, which cannot be done while executing $add).

Icon Colors

The $iconcolor and $defaulticoncolor properties control the color of icons when using themed SVG icons. The **$iconcolor** property for a remote menu line sets the icon color when using a themed SVG icon. The **$defaulticoncolor** property for a remote menu class sets the icon color when using a themed SVG icon *and the $iconcolor property of the item is kColorDefault*. If $defaulticoncolor is also kColorDefault, then the themed icon uses the text color.

Text Alignment

You can change the text alignment in Remote menus. The pContextMenu event parameter in evOpenContextMenu events has an $align property. This can be used to specify the text alignment of a menu. For example:

Do pContextMenu.$align.$assign(kCenterJst)

Possible values are kLeftJst (default), kRightJst and kCenterJst.

Context Menus

A *context menu* is a menu that can be opened by the end user by right-clicking on the background of a form, or within the border of a form control. You can implement context menus for remote forms or individual form controls by setting the $contextmenu property of the form or control to the name of a Remote menu class. Each line in a remote menu has the $commandid property, so when the user selects a line in the menu this ID can be used to trigger a specific action in your code. When a line in a remote menu is selected, an evExecuteContextMenu event is reported to the form or field with event parameters containing the Command ID of the selected line, and for fields, a reference to the field which was clicked on.

Remote forms and controls have the $disabledefaultcontextmenu property to disable default menus from opening when the end user right-clicks the form or object: the default menu for the edit control could be the clipboard menu. If true, the default context menu for the object will not be generated in response to a context click ($clib.$disabledefaultcontextmenu and $cobj.$disabledefaultcontextmenu must both be false for the default menu to be generated).

Context Menu Events

Remote forms or controls report the following events in response to a context click.

- **evOpenContextMenu**
  Sent to a field or a remote form when a context menu is about to open; the event contains the parameters:
  pContextMenu is a reference to the remote menu instance that is about to pop up as a context menu;
  pControlMenu is kTrue if the menu is a control menu, or kFalse if it is a context menu, see below;
  pClickedField is a reference to the field which was clicked.
  For data grids only, the pPosition parameter for the evOpenContextMenu event contains a co-ordinate string, such as “4,2”, where the first number is the row and the second is the column: the column part will always be populated with the column you right-clicked under, but the row part will only be non-zero if you right-click on a row

- **evExecuteContextMenu**
  Sent to a field or remote form when a context menu item is selected; the event contains the parameters:
  pCommandID is the command ID of the selected remote menu item;
  pControlMenu is kTrue if the menu is a control menu, or kFalse if it is a context menu, see below;
  pClickedField is a reference to the field which was clicked

The remote menu instance itself only exists during the evOpenContextMenu event, therefore it is possible to modify the menu before it is displayed on the client, or you can discard the event to prevent the menu from being displayed.

Remote form instances have the property $remotemenu which is the name of the remote menu instance (set only when evOpenContextMenu for the field or form is being processed): for hierarchical menus, this is the item reference to the remote menu instance of the attached remote menu.

After evOpenContextMenu completes, and the user selects a remote menu item, the client sends an evExecuteContextMenu event to the form or form control that received the evOpenContextMenu, passing the event parameter pCommandID containing the value of $commandid of the selected menu line.

Control Menus

All controls with a menu, such as Tab, Popup menu, or Split button, generate the evOpenContextMenu and evExecuteContextMenu events when using their own control menus. The pControlMenu parameter can be used to distinguish between Control menus and Context menus; it is kTrue if the menu is a Control menu, or kFalse if it is a context menu.

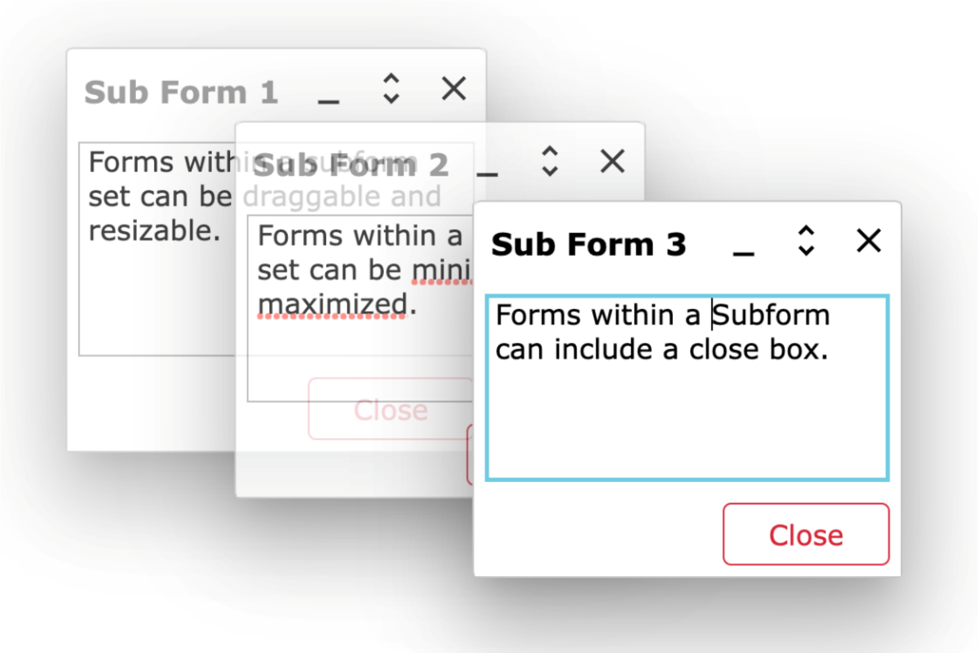
Subform Sets

You can open a special kind of subform or group of subforms that behave like separate “floating” windows inside the main remote form in the JavaScript Client. The subforms in a **Subform Set** (SFS) are different to standard subforms (which are embedded in a Subform control, and described in the *JavaScript Components* chapter), in that they have a title bar and resizable borders, and the end user can move or resize them within the “main” or parent remote form instance running on the client. Such dynamic subforms within a subform set allow you to create highly flexible user interfaces in your apps, by allowing a high degree of interactivity for the end user.

The subforms in a subform set are *created at runtime* in the JavaScript Client within a remote form instance, or they can be opened within the context of a single page in a paged pane in a remote form, which are referred to as Subform panels. Each separate subform in the Subform Set is an instance of a standard Remote form class that you have previously created in your library, which is referenced and added to the Subform set on the client.

In addition to opening one or more subforms in Subform set, you can open a subform as a Subform Palette; see Subform Palettes.

There is an example app in the **Samples** section in the **Hub** in the Studio Browser showing how you can setup and use a subform set, and the same app is available in the JavaScript Component Gallery.



Forms with
set can be
resizable.

draggable and

Forms within a
set can be mini
maximized.

Forms within a Subform
can include a close box.

Figure 94:

Subform set windows respect their container’s dimensions when opened, that is, if subforms that are larger than the available space are opened (or are too far to the right or bottom to fit the whole subform area), then they will be resized and repositioned automatically.

Specifically, the left and/or top values will be reduced, If these reach 0, and there is still not enough space for the subform, the width and/or height values will also be reduced.

Stacking Order List

The subforms in a Subform Set have a "stacking order" (or Z-order) relative to one another, so the top-most form in the set will appear in front of any forms lower in the stacking order if they intersect. Clicking on a form lower in the stacking order brings it to the front. The tab order of the remote form excludes controls on the subforms in a Subform Set behind the top form in the set. There is a maximum of 256 remote form instances (including subform instances) in a remote task instance. When you have more than one Subform Set open (this is allowed but not recommended) there is no relative stacking order between the sets.

Creating Dynamic Subforms

There is a set of *client commands* to open and manage the subforms in a Subform Set which you can execute using the $clientcommand() method. These client commands should be executed in the context of the current remote form instance using $cinst. The $clientcommand() method requires two parameters: the *cCommand* to be executed and a *wRow* variable containing the parameters for the command, with the syntax:

```
Do $cinst.$clientcommand(cCommand,wRow)
```

where $cinst is the current remote form instance.

Subform Client Commands

The following client commands are available for creating and managing subform sets or the subforms in a subform set, including subform dialogs.

subformset_add

The **subformset_add** command creates a Subform Set within the current remote form instance.

```
Do $cinst.$clientcommand("subformset_add",rowVariable)
```

Where *rowVariable* is row(**setname, parent, flags, ordervar, formlist**)

Note the parent parameter is available if you want to create the subform set inside a paged pane, rather than the remote form instance. The columns for the row variable parameter are as follows:

setname: a string which is the name of the subform set, which must be unique within the current remote task.

parent: the container for the set, either:

- pagedpanename:page (e.g. pp:5), so that the subforms belong to the specified page of the paged pane)
- or empty, meaning that the subforms in the set belong to the remote form instance invoking $clientcommand.

flags: the sum of the following constants (e.g. kSFSflagMinButton+kSFSflagMaxButton), which effect the behavior of the subforms in the set:

- **kSFSflagCloseButton:** The subforms in the set have a close button
- **kSFSflagMinButton:** The subforms in the set have a minimize button
- **kSFSflagMaxButton:** The subforms in the set have a maximize button; note the subform can only be maximized (resized) if kSFSflagResize is enabled
- **kSFSflagResize:** The subforms in the set have a resize border so that they can be resized using the mouse or when the Maximize button is pressed

- **kSFSflagOpenMin, kSFSflagOpenMax, kSFSflagMinAsTitle, kSFSflagAutoLayout: are used to open a subform set within a “panel” or container, such as a paged pane control: see Subform Panels later in this section
- **kSFSflagPreventDrag:** the user will not be able to drag the subforms in the SFS
- **kSFSflagScrollable:** allows subform sets to scroll; only effects non-responsive subform sets since responsive subform sets are scrollable by default
- **kSFSflagAllowOutsideOfBounds:** allows subforms to be positioned outside of their container boundaries, both by notation and the end user dragging them
- **kSFSflagEscToClose:** the subforms in the set can be closed by pressing the Escape key

ordervar: the name of an instance list variable in the remote form invoking the $clientcommand. The client keeps this list variable up to date with the stacking order and position information for the subforms in the set: see below.

formlist: a list which defines the subforms to be added initially to the subform set (this list can be empty), i.e. a list of remote form classes that you have previously created in your library. The order of the forms in this list represents the stacking order from top to bottom, so that once the set has been added, the top-most subform will be for line 1, and the bottom-most subform will be for the last line. The columns in the list are as follows:

- Column 1: uniqueID: An integer which must uniquely identify this subform in the set.
- Column 2: classname: The name of the Omnis remote form class for the subform.
- Column 3: params: Literal parameters to be passed to the $construct and $init of the subform, e.g. “ ‘Test’,200”
- Column 4: title: The title of the subform - text displayed in the title bar of the subform.
- Column 5: left: The left coordinate of the subform (for Desktop browsers, or portrait if a mobile device). The constant kSFScenter centers the form horizontally in its parent.
- Column 6: top: The top coordinate of the subform (for Desktop browsers, or portrait if a mobile device). The constant kSFScenter centers the form vertically in its parent.
- Column 7: width: The width of the subform. If the forms in the set are resizable, then the form cannot be made narrower than the minimum of this width and the width designed for the remote form class.
- Column 8: height: The height of the subform. If the forms in the set are resizable, then the form cannot be made taller than the minimum of this height and the height designed for the remote form class.
- If the subforms are to be displayed on a mobile device, Columns 9-12 are landscape left, top, width and height respectively. If these are omitted, the landscape values default to the portrait values.

When formLeft or formTop parameters are set to kSFScenter, the subform will be displayed in the center of the current viewport or the current form, whichever is the smaller of the two: note that horizontal and vertical centring work independently of each other.

Formlist for Responsive forms

When using the “subformset_add” and “subformset_formadd” client commands you can pass dimensions for the subforms *for each breakpoint* in a responsive remote form in the formlist parameter (instead of the single set of left, top, width, height parameters). The same method applies to both client commands, but the example below shows directly adding a form with “subformset_formadd”:

```
Do lDimList.$define(lBreakpoint,lLeft,lTop,lWidth,lHeight)
Do lDimList.$add(310,10,10,200,200)
Do lDimList.$add(600,kSFScenter,kSFScenter,300,300)
Do lDimList.$add(1000,kSFScenter,kSFScenter,600,600)
Do $cinst.$clientcommand( "subformset_formadd",row(cSetName,vUniqueID,cParams,cTitle,lDimList,iModal))
```

The dimensions list replaces the 4 separate parameters, and so condenses the command to a minimum of 5 parameters (6 if passing a value for iModal): this only works for responsive forms, whereas single and screen type forms must use the original set of parameters to avoid confusion.

The client will use the value passed in lBreakpoint to assign the values to the correct breakpoint for the containing form. If the breakpoints do not match, then the values will be used from the next breakpoint down. For example, if you had the list of dimensions as defined above, but your form used the following breakpoints:

310 - would use the values from 310 as they match

590 - this is smaller than the next value of 600, so would again use the values from 310

900 - this is smaller than the next value of 1000, so would use the values from 600

1200 - this is greater than the values from 1000, so uses those values.

subformset_formadd

Having created a subform set using **subformset_add,** you can use the **subformset_formadd** client command to add a form to the subform set.

```
Do $cinst.$clientcommand("subformset_formadd",rowVariable)
```

Where *rowVariable* is row(**setname, uniqueID, classname, params, title, left, top, width, height, modal**)

The row variable parameter are as follows:

setname: a string which is the name of the set to which the subform is to be added.

uniqueID: an integer which must uniquely identify this subform in the set.

classname: the name of the Omnis remote form class for the subform.

params: A comma-separated list containing literal parameters to be passed to the $construct and $init methods of the subform instance, e.g. " 'Test',123". Note strings have to be quoted, and can contain spaces and commas. $init is run in serverless client subforms.

title: the title of the subform - text displayed in the title bar of the subform.

left: the left coordinate of the subform (for Desktop browsers, or portrait if a mobile device). The constant kSFScenter centers the form horizontally in its parent.

top: the top coordinate of the subform (for Desktop browsers, or portrait if a mobile device). The constant kSFScenter centers the form vertically in its parent.

width: the width of the subform. If the forms in the set are resizable, then the form cannot be made narrower than the minimum of this width and the width designed for the remote form class.

height: the height of the subform. If the forms in the set are resizable, then the form cannot be made taller than the minimum of this height and the height designed for the remote form class.

Note the 4 separate parameters (left, top, width, height) can be replaced by a list containing the dimensions for each breakpoint: see above regarding the formlist for subformset_add.

modal: zero if the subform is non-modal, or 1 if the subform is fully modal, and prevents the use of any other form or subform in the remote task's user interface (the form is grayed out and clicks outside the subform are ignored).

If the subforms are to be displayed on a mobile device, the next four columns are landscape left, top, width and height respectively. If these are omitted, the landscape values default to the portrait values.

The parameters above, starting with uniqueID, are identical to those in the formlist (for the subformset_add command), except the modal indicator is present between the two sets of coordinates.

subformset_remove

The **subformset_remove** command removes a set of subforms. All subforms in the set will be destructed and removed from their parent.

```
Do $cinst.$clientcommand("subformset_remove",rowVariable)
```

Where *rowVariable* is row(**setname**) where **setname** is set to be removed.

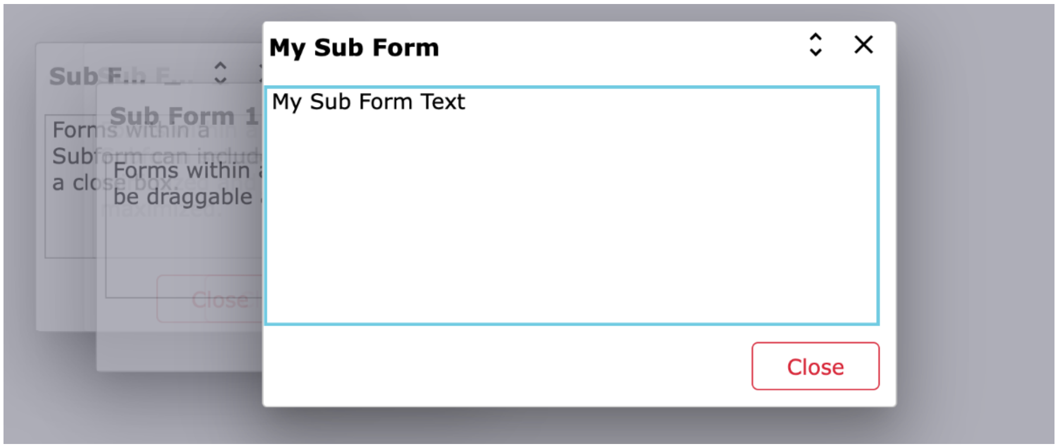


Figure 95:

subformset_formremove

The **subformset_formremove** command removes a subform from an existing set and destructs it (removing it from its parent).

```
Do $cinst.$clientcommand("subformset_formremove",rowVariable)
```

Where *rowVariable* is row(**setname, uniqueID, focus**)

The row variable parameter are as follows:

setname: a string which is the name of the set from which the subform is to be removed.

uniqueID: an integer which identifies the subform in the set to be removed.

focus: optional (default value is kFalse). If the focus parameter is kTrue, sets focus to the new top form in the set unless it is minimized.

subformset_formtofront

The **subformset_formtofront** command brings a subform in a set to the top of the stacking order, and gives it the focus. You must use this command to display a subform that has previously been minimized.

```
Do $cinst.$clientcommand("subformset_formtofront",rowVariable)
```

Where *rowVariable* is row(**setname, uniqueID**)

setname: a string which is the name of the set containing the subform.

uniqueID: an integer which identifies the subform in the set to be brought to the front.

Subform Dialogs

subformdialogshow

The **subformdialogshow** command opens a single subform as a modal dialog.

```
Do $cinst.$clientcommand("subformdialogshow ", rowVariable)
```

Where rowVariable is row(classname, params, title, width, height, closeButton, resizable, maxButton, openMax). The parameters are as follows:

Parameter	Description
classname	String, the name of the remoteform
params	String literals to pass to the subform

Parameter	Description
title	String, the title of the modal dialog
width	Integer, the width of the dialog
height	Integer, the height of the dialog
closeButton	Boolean (Optional), defaults to true, show close button
resizable	Boolean (Optional), defaults to false, if true allows resizing
maxButton	Boolean (Optional), defaults to false, if true shows maximize button (resizable must be set to true)
openMax	Boolean (Optional), defaults to false, if true opens dialog in a maximized state (resizable must be set to true)

This command generates a new subform set and adds one modal subform to it. The name of this set is internal only, and cannot be added to or removed from. Another modal subform dialog can be opened above the previous one by running subsequent calls, preventing access to the first one until the second is closed (using the subformdialogclose command).

subformdialogclose

The **subformdialogclose** command closes the topmost subform dialog. This command only works for subforms opened using the subformdialogshow command, and has no parameters as such modal dialogs must be closed in reverse order of them opening.

Using the Stacking Order Variable (ordervar)

When you use the subformset_add client command a list called ordervar is created containing a list of the subforms in the subform set. The ordervar variable allows you to manage the subforms in the set. It has the same definition as the formlist, and like the formlist it contains the subforms in the order of the top to the bottom of the stacking order. Note that if coordinates have been centered using kSFScenter, the ordervar contains their actual values rather than the value kSFScenter.

Whenever the stacking order changes, or a form is moved or resized, the client updates the values in ordervar. This results in:

- Automatic updates to controls which are data-bound to the ordervar.
- A call to a client method in the container form for the SFS. If you add a client-executed method called $sfsorder, with a single parameter, which is the set name, you can add processing that occurs each time the set is updated. For example, you could use a tab control to display a tab for each member in the set, where the current tab represents the top-most subform.

You can use the ordervar list in conjunction with the subformset_formtofront $clientcommand to manage the subforms in the set, e.g. bring a form to the front by selecting a line in a popup menu (this is the only way to restore a minimized subform). For example:

```
# the ordervar list is assigned to iOpenForms, C1 contains the subform ID
Do $cinst.$clientcommand("subformset_formtofront",row('SubformSet',iOpenForms.[pLineNumber].C1))
```

If a subform has been minimized, you would you have to use such a method to display the subform again since minimized subforms are not visible in the parent remote form.

Load Finished Method

Remote forms have the $loadfinished client-executed method which is called after all the subforms that belong to the parent remote form instance have finished loading and their $init methods have been called; so you could create a client method called $loadfinished to perform any actions you want after all subforms have loaded.

Trapping the Close event

You can add a method to a subform in a subform set to trap the close event before the $destruct for the subform is run. The method should be named $sfscanclose and should be set to be client-executed (which should be enabled automatically). The method can contain whatever processing you want to run. If the return value (a Boolean) from this method is kFalse the subform cannot close. Otherwise, if the return value from this method is kTrue, the subform can close. If $sfscanclose is not present, the subform closes by default.

Dialog and Palette style subforms can call into $sfscanclose() when attempting to close via the close button (X) in a dialog subform, or clicking the background outside the palette for a palette style subform. As with other subforms, it would be possible to test a conditional statement in $sfscanclose() and if it returns kFalse the close event will be cancelled.

Scroll Position

The kSFSflagPosnScroll flag allows you to control how a subform is positioned relative to its container if the container has been scrolled. When the kSFSflagPosnScroll is set the subform in a subform set (SFS) will open relative to the current scroll position of its container. For example, on a long form which is currently scrolled to the bottom of the page, with a subform opening at left position 100 and top position 100, it will open 100 pixels in from the top of what can currently be seen in the viewport. Similar behavior would apply if it belonged to a paged pane that has been scrolled.

When the flag is not set, it will be positioned absolutely to the defined position: therefore, in a long form that has been scrolled to the bottom of the page, the subform will be placed at the top of the page if its top position is set to 0.

However, when opening a modal subform in a subform set, if its position is set to kSFSCenter, it will always be positioned relative to the current scroll position of the form, as modal subforms always belong to the form, not a paged pane (since a modal subform requires interaction and closing before any other action can be taken on the form). If kSFSCenter and kSFSflagPosnScroll are both not used, then a subform will be placed at its specified position, even if that is out of the current view of the user (which is the behavior in previous versions).

Subform Styles

The styles and colors used in subforms will be those set in the subform classes; the colors for the subform set are taken from the theme used in the main JS form.

Subform Titles

You can use $cinst.$title to change the title text for a member of a subform set.

Subform References

You can obtain a reference to any subform instance within a subform set using the $sfsmember root notation. For example:

```
$root.$sfsmember(cSetname,iUniqueID)
```

returns an item reference to the remote form instance for the subform set member with the specified unique ID in the named subform set in the current remote task. This notation can be used in server and client methods.

Subform Panels

You can open a set of subforms as a group of collapsible panels within a container, such as a paged pane control. The subform panels are arranged vertically and the end user can expand and collapse each subform by clicking or tapping on the subform title bar or the minimize icon. Subform panels cannot be nested.

There is an example app called **JS Subform Set Panels** in the **Samples** section in the **Hub** in the Studio Browser showing how you can setup and use subform panels, and the same app is available in the JavaScript Component Gallery.

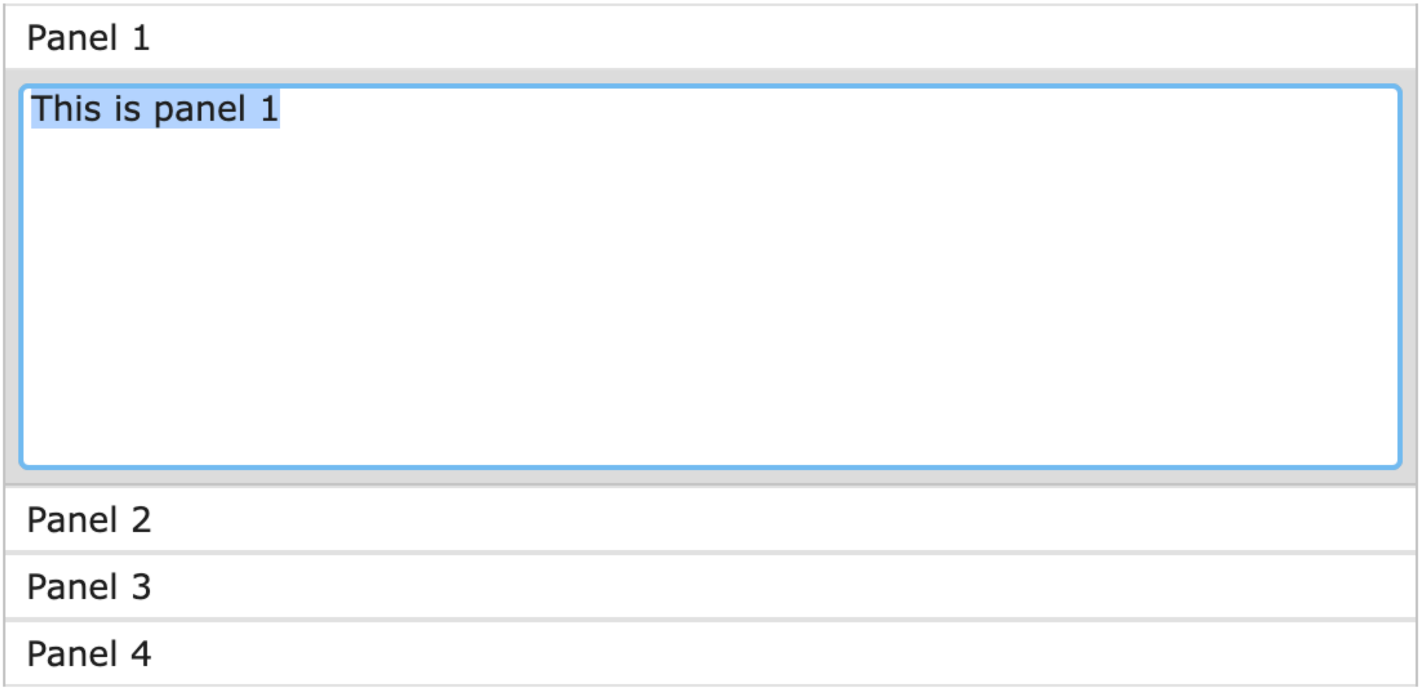


Figure 96:

Configuring the panels

You can create a set of subform panels using the “subformset_add” client command along with the “kSFSflag…” constants, which can be found in the Catalog (F9) in the ‘Subform sets’ group. The subformset_add command creates a set of subforms within the current remote form instance or a parent, such as a paged pane.

```
Do $cinst.$clientcommand("subformset_add",rowVariable)
```

where *rowVariable* is row(setname, parent, flags, ordervar, formlist). The flags can be summed to specify the complete behavior of the panels in the set: see the example method below to see how to use subformset_add and the flags.

kSFSflagOpenMin

The subform panels in the set are opened in the minimized state. Normally, all subforms in the set are opened in the un-minimized state. This flag overrides this default behavior.

kSFSflagOpenMax

The subform in the set is opened in the maximised state. Sizes/positions should still be set as the subform will return to these values if it is restored.

kSFSflagMinAsTitle

When a panel (subform) in the set is minimized, just the title bar is shown. This flag overrides the default behavior which is to reduce the subform to nothing when it is minimized.

You can use the kSFSflagMinButton flag to add a minimize button to each subform to allow the end user to expand and contract the panel (in addition to clicking on the title).

kSFSflagAutoLayout

Automatically lay out the panels (subform set members) vertically within their parent, ignoring the specified left and top. Turns on kSFSflagMinAsTitle and turns off kSFSflagResize and kSFSflagMaxButton. When using kSFSflagAutoLayout, the user can drag and drop the title bar of the panels in the set, to re-order them. If you open a modal subform in a set with kSFSflagAutoLayout set, the modal form opens at the top of the parent form, and does not become part of the vertically laid out forms.

kSFSflagParentWidth

Only applies when kSFSflagAutoLayout is specified. Ignores the width parameter for each set member, and sets the width of each subform to the width of parent. This flag also sets edgefloat for each subform to kEFright. Using kSFSflagParentWidth allows you, for example, to create a paged pane page populated with panels implemented as subforms, where the panels resize when the paged pane resizes.

kSFSflagSingleOpen

Only applies when kSFSflagAutoLayout and kSFSflagMinButton are both specified. When specified ensures that at least one window is always open.

kSFSflagMinButtonIsTitle

The minimize button icon is removed and the whole title bar becomes the minimize button. Only applies when kSFSflagAutoLayout is specified.

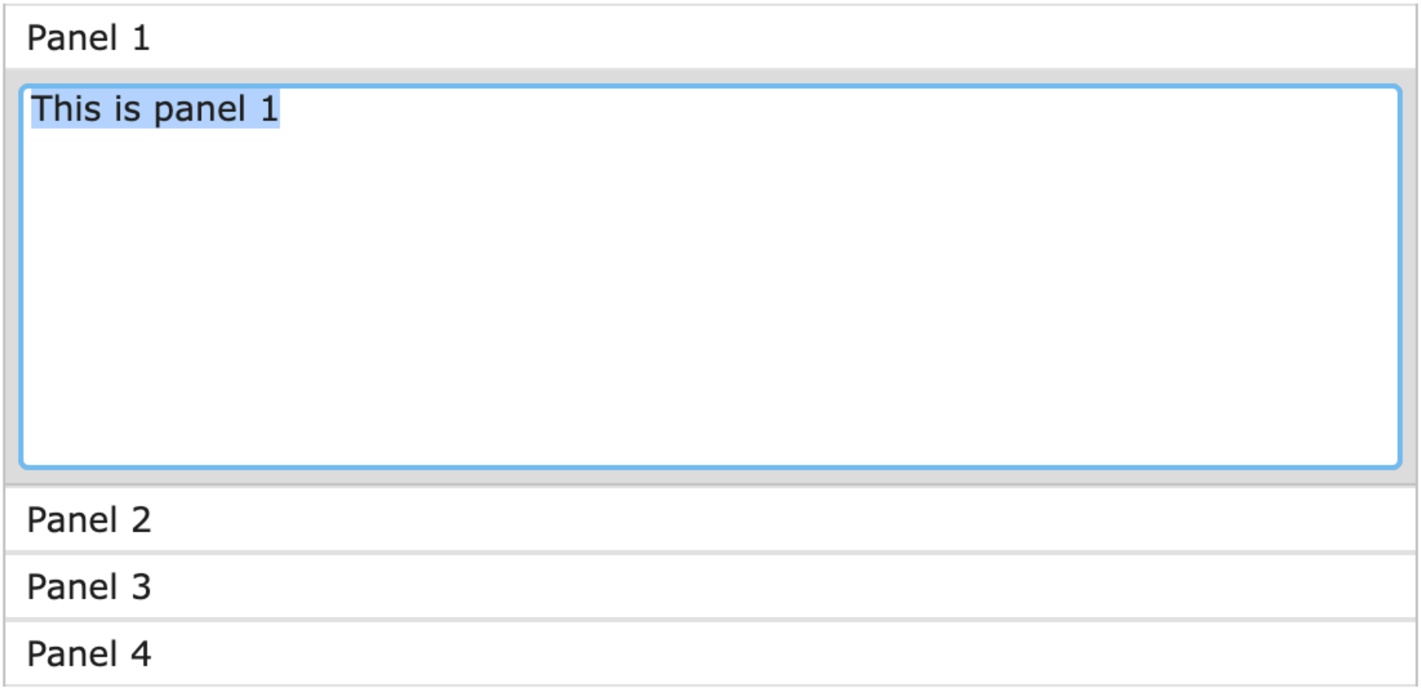
Expanding and Collapsing Subform panels

End users can expand or collapse (open or close) subform panels using a single click (in previous versions, a double-click was required). This is enabled by making the title bar on the subform behave like the minimize button, and therefore the title accepts single clicks. The **kSFSflagMinButtonIsTitle** flag for the ‘subformset_add’ action needs to be set to allow this behavior, and only applies when kSFSflagAutoLayout is specified.

When in auto layout mode, and not using single open or open minimized modes, you can indicate that a form is to be opened minimized modes by prefixing its class name with the ~ (tilde) character. This means that when you open a number of subforms in a subform set, you can specify which subforms will open minimized (collapsed).

Subform Panels Example

The following example of a set of subform panels contained in a paged pane (available in the Samples section in the Hub), with both the auto layout and parent width flags set.



Panel 1

This is panel 1

Panel 2

Panel 3

Panel 4

Figure 97:

The following method constructs the subform set and assigns it to a paged pane in the remote form. In this case, the subform only contains an edit control which receives some text to be displayed in the subform (“This is panel #”). The list of subforms is built including the text and background color which is assigned to a paged pane using a row variable and the subformset_add client command.

```
# create list vars lFormList and lSetRow and all columns
# create the list of subforms in lFormList
Do lFormList.$define(lFormID,lClassName,lFormParams,lFormTitle,lFormLeft,lFormTop,lFormWidth,lFormHeight)
Do lFormList.$add(1,'jsSubformSetPanelsSubForm',con(1,chr(44),rgb(221,221,255),chr(44),chr(34),"This is panel
Do lFormList.$add(2,'jsSubformSetPanelsSubForm',con(2,chr(44),rgb(204,204,255),chr(44),chr(34),"This is panel
Do lFormList.$add(3,'jsSubformSetPanelsSubForm',con(3,chr(44),rgb(187,187,255),chr(44),chr(34),"This is panel
Do lFormList.$add(4,'jsSubformSetPanelsSubForm',con(4,chr(44),rgb(170,170,255),chr(44),chr(34),"This is panel
# construct the row for the subformset_add command in lSetRow
Do lSetRow.$define(lSetName,lParent,lFlags,lOrderVar,lFormList)
Do lSetRow.$assigncols("SubformPanelsSet",'PagedPane:1',kSFSflagSingleOpen+kSFSflagMinButton+kSFSflagAutoLayou
+kSFSflagParentWidth,'iOpenForms',lFormList)
Do $cinst.$clientcommand("subformset_add",lSetRow)
```

In this case the flags kSFSflagSingleOpen, kSFSflagMinButton, kSFSflagAutoLayout, and kSFSflagParentWidth have been summed to create the complete properties for the set of panels.

An example containing a set of Subform Panels is available in the JavaScript Components Gallery on the Omnis website.

Modal Subforms and Paged Panes

If you associate a subformset with a paged pane and you add a new MODAL form to the set, the new subform window will be associated with the remote form as a whole, rather than the paged pane.

Subform Set Example

The following code creates a subform set containing two subforms.

```
# the following setupSubformSet method could be called from $construct
# Create vars: iFormList (List), iID, iClassName, iParams, iTitle, iLeft, iTop, iWidth, iHeight
# Create local vars in setupSubformSet: lRow (Row), lSetName, lParent, lFlags, lOrderVar
# jsSub1 and jsSub2 are remote forms in the library
Do iFormList.$define(iID,iClassName,iParams,iTitle,iLeft,iTop,iWidth,iHeight)
Do iFormList.$add(1,"jsSub1",,"subform1",10,10,200,200)
Do iFormList.$add(2,"jsSub2",,"subform2",220,10,400,200)
Do lRow.$define(lSetName,lParent,lFlags,lOrderVar,iFormList)
Do lRow.$assigncols("SubformSet",,kSFSflagCloseButton+kSFSflagMaxButton + kSFSflagMinButton+kSFSflagResize,,iF
Do $cinst.$clientcommand("subformset_add",lRow)
```

The code creates the subforms in the main remote form within the browser:

The **Memo** sample app available in the **Applets** section of the **Hub** uses subform sets.

Using Subform Palettes

A *Subform Palette* is a subform that can be opened next to a specified control. Such a subform could allow the end user to set an option, or to provide some information such as a help tip. Subform palettes can be opened or closed using the client commands **subformpaletteshow** and **subformpaletteclose,** or a subform palette can be closed by clicking outside the subform.

There is an example application called **JS Subform Dialogs** in the **Samples** section of the **Hub** in the Studio Browser showing how to use the subform palette client commands.

Showing a Subform Palette

The **subformpaletteshow** command shows a remote form as a subform palette:

```
Do $cinst.$clientcommand("subformpaletteshow",rowVariable)
```

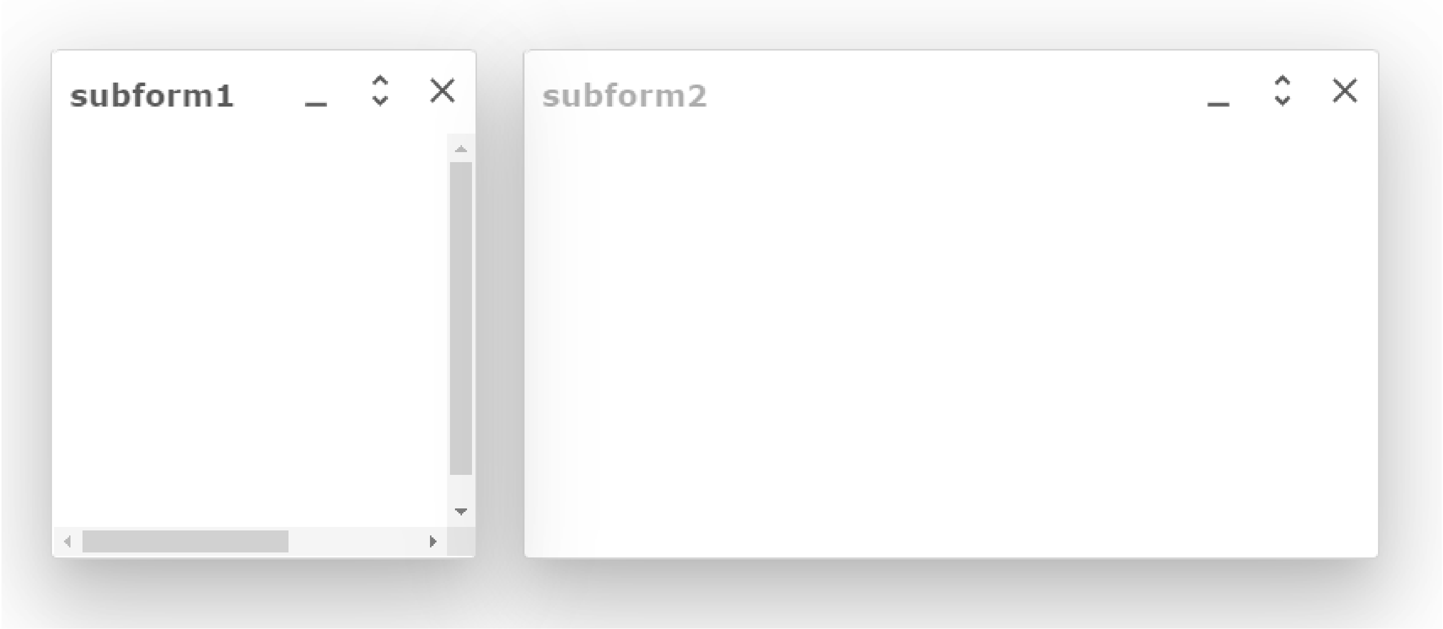
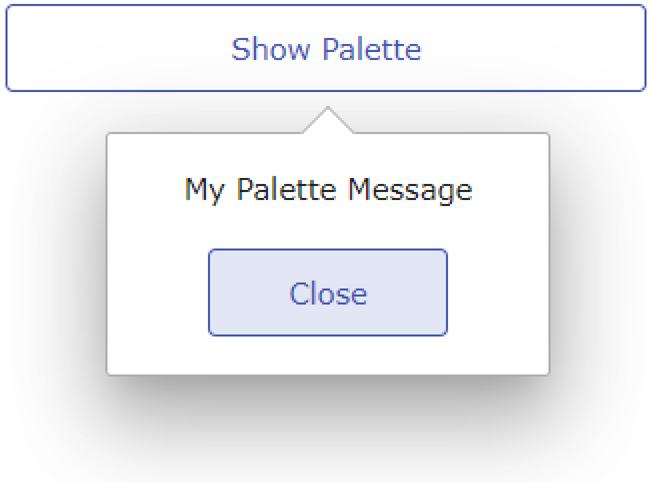


Figure 98:



My Palette Message

Figure 99:

Where rowVariable is row(cClass, cParams, cControl, iWidth, iHeight [,iPositionFlags] [,bShowOverlay] [,cTitle] [,bPreventClose])

The row variable parameters are as follows:

Parameter	Description
cClass	Character, the class name of the remote form to use in the palette.
cParams	Character, parameters to pass to the remote form, e.g. you could pass a message (text) to be displayed in the subform palette, as in the example app.
cControl	Character, the name of the related control to pop up the palette next to.
iWidth	Integer, the width of the subform palette.
iHeight	Integer, the height of the subform palette.
iPositionFlags	Integer (Optional), a combination of up to 2 kSFSPalette… constants to specify the position (defaults to kSFSPalettePosFlagRight+kSFSPaletteAlignFlagCenter). See further description of flags below.
bShowOverlay	Boolean (Optional), if kTrue, shows the form overlay while the palette is open (defaults to kFalse).
cTitle	Character (Optional), the title of the subform (this is not displayed anywhere, but is used to populate the aria-label property of the palette for screen readers).
bPreventClose	Boolean (Optional), if kTrue, the user cannot close the palette by clicking outside it (defaults to kFalse); in this case, you must use **subformpaletteclose** to close the palette.

The **Positioning flags** should contain up to 1 Position flag (top, right, bottom, or left) and 1 Align flag (start, center, or end) to set the position of the subform palette relative to the control specified in cControl.

Constant	Description
kSFSPalettePosFlagTop	Position the palette above the related control
kSFSPalettePosFlagRight	Position the palette to the right of the related control
kSFSPalettePosFlagBottom	Position the palette below the related control
kSFSPalettePosFlagLeft	Position the palette to the left of the related control
kSFSPaletteAlignFlagStart	Aligns the palette at the start of the specified position (top/right/bottom/left)
kSFSPaletteAlignFlagCenter	Aligns the palette in the center of the specified position (top/right/bottom/left)
kSFSPaletteAlignFlagEnd	Aligns the palette at the end of the specified position (top/right/bottom/left)

If the palette were to overlap the opposite side of the control, e.g. because of the lack of space, Omnis will try to place the subform on a different edge automatically. If Omnis cannot place the subform in an acceptable position, it will fallback to using the initial state.

There is also an arrow which will point to the center of the given control. However, it is restricted by the size of the subform palette, and so will be placed towards the edge of the palette closest to the center of the control, as appropriate.

In the example app (in the Hub), the following code is in the $event method for the button which opens a subform palette next to the button (iPaletteMessage is populated with a text message from the form, while iPalettePosValue and iPaletteAlignValue are taken from droplists in the form):

```
Do $cinst.$clientcommand("subformpaletteshow",row("jsSFDPalette",con(kDq,iPaletteMessage,kDq),"ShowPalette",22
iPalettePosValue+iPaletteAlignValue,kFalse,"MyTitle"))
```

Closing a Subform Palette

The **subformpaletteclose** client command closes the top-most subform palette. No row parameter is required. Note the end user can close a subform palette by clicking outside the subform (which can be prevented using bPreventClose).

Running JavaScript in the Client

The JavaScript: command (including the colon) allows you to execute any native JavaScript in the client browser directly from your Omnis code; this can include calls to other JavaScript embedded into or linked to the HTML page containing your JavaScript remote form. Omnis does not perform any validation of the code you insert into the JavaScript: command in the method editor (you can check for errors in the JavaScript console of the browser). You cannot include any square bracket notation in the code parameter of the JavaScript: command, since the code needs to be evaluated by the server when generating the script file. A simple example would be to open an alert in the browser with the standard alert() function, as follows:

```
JavaScript:alert("Hello World");
```

Since this command executes native JavaScript code on the client it must be executed in a client-side method. The JavaScript: command will only appear in the Command list in a method that is enabled to execute on the client. Omnis will not allow the JavaScript: command to be present in a server method (an error occurs). If you try to execute JavaScript: on the server, by calling a client-side method from a server method, a debugger error will occur.

You can include JavaScript in the HTML page containing your remote form (inline or linked), and call code in these scripts from your remote form code using the JavaScript: command. To extend the alert() example above, you could embed a JavaScript function in your HTML page, like this:

```
<html>
  <head>
    <script type="text/javascript">
    function show_alert()
    {
      alert("Hello World");
    }
    </script>
  </head>
  <body>
    <!-- body including the omnisobject containing your form -->
  </body>
</html>
```

And call this function from your Omnis code using the JavaScript: command as follows:

```
JavaScript:show_alert();
```

Another example could include using the JavaScript: command to “push” events to Google Analytics to track certain actions or events in your application. To do this you would need to add the standard Web Analytics code provided by Google into your HTML page (containing your remote form) and call the gaq.push() function with the correct parameters from within your Omnis code.

Example Using the JavaScript: Command

A possible use for the JavaScript: command is for an event handler. You could use the $init method to install an event handler, such as:

```
JavaScript:document.getElementById("jsTEMP1_1076_client").onselect=function(event) { __form.callMethodEx("sele
JavaScript:window.addEventListener("keydown",function(event){ __form.callMethodEx("keydown",0,event) }, true);
```

Note that in both of these examples you can use __form.callMethodEx to call an instance method of the remote form. The second parameter (zero in these examples) are flags which control how callMethodEx behaves – these must always be passed as zero.

You could use the $init method to assign additional style information to controls on the form. For example:

```
Do $cinst.$objs.$sendall($cinst.$addboxshadow($ref))
```

$addboxshadow is a form method with parameter pObj. The following JavaScript: command adds a box shadow if the browser supports it.

```
JavaScript:pObj.elem.style.boxShadow="0px 0px 5px 5px #888888";
```

Styled Text

You can insert various text styles in some of the JavaScript components wherever text is displayed. For example, you can insert colors, font styles, and images into the text within the list control, the droplist control, the data grid control, and the hyperlink control.

Text styles can be inserted using the style() function, in both server and client methods. The style() function inserts a style-character represented by an Omnis constant into a calculation or a text block. The styles that can be used include some of the existing constants (listed under ‘Text Escapes’ in the Omnis Catalog) and a few new ones, prefixed kEscJs..., introduced for the JavaScript Client. The following style constants can be used:

Style	Description
kEscColor	In a client-side method, the color parameter can be a numeric literal, a constant such as kRed, or HTML color string enclosed in double quotes.
kEscStyle	In a client-side method, the style parameter must be a numeric literal or constant.
kEscBmp	In a client-side method, the icon id parameter must be either a numeric literal, or the sum of a numeric literal and an icon size constant e.g. 1710+k48x48
kEscJsNewline	No additional parameters are required. Inserts a line break tag. See *
kEscJsClose	No additional parameters are required. Closes the current open style information (inserted by kEscColor or kEscStyle) in the styled text and reverts to the original color and text style. Note - kEscColor and kEscStyle insert a tag to style the text. kEscJsClose closes the tag.
kEscJsHtml	Inserts raw HTML (the second parameter to style()).

* You can use the br() function as a short-hand to insert a new line. The br() function and kEscJsNewline can only be used with styled text for the JavaScript client.

The parameters for style() can use any HTML color string, such as “#FF0000”. For client methods that execute on the client, the color parameter *must be a literal string* and therefore enclosed in double quotes. For example, style(kEscColor,“#FF0000”), or

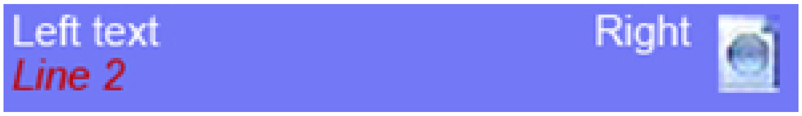


Figure 100:

style(kEscColor,“rgba(0,0,255,0.5)”). Omnis does not validate the HTML color syntax, so you should check the syntax is correct to avoid runtime errors.

The following example code produces a list line which looks like this from the styled text data in iChar:

```
Calculate iFloatRight as style(kEscJsHtml,"<span style='float:right;'>")
Calculate iFloatLeft as style(kEscJsHtml,"<span style='float:left'>")
Calculate iSmallFont as style(kEscJsHtml,"<span style='font-size:8pt'>")
Calculate iCloseSpan as style(kEscJsHtml,"</span>")
Calculate iIcon as style(kEscBmp,1710+k32x32)
Calculate iOpenP as style(kEscJsHtml,"<p style='height:36px;margin:0px'>")
Calculate iCloseP as style(kEscJsHtml,"</p>")
Begin text block
  Text: [iOpenP][iFloatLeft]Left text[iCloseSpan]
  [iFloatRight]Right[iIcon][iCloseSpan][br()]
  [iSmallFont][style(kEscStyle,kItalic)]
  [style(kEscColor,kRed)]Line 2[iCloseSpan][iCloseP]
End text block
Get text block iChar
```

Animations

JavaScript remote forms have the methods $beginanimations() and $commitanimations() which allow you to control animations for some controls. The animated properties are: left, top, width, height, alpha, backcolor, backalpha, textcolor, fontsize, bordercolor, linestyle, buttoncolor (the latter is for pushbutton only).

There is a sample app called **Animations** in the JavaScript Component Gallery, and under the **Samples** option in the **Hub** in the Studio Browser showing how you can animate component changes such as size, position, and transparency.

The syntax for the $beginanimations() method is:

- $beginanimations(iDuration[,iCurve=kJSAnimationCurveEaseInOut])
  after calling this, assignments to some control properties are animated for iDuration (milliseconds) by executing $commitanimations()
  iCurve values are:
  kJSAnimationCurveEaseInOut (the default), kJSAnimationCurveEaseIn, kJSAnimationCurveEaseOut, kJSAnimationCurveEase and kJSAnimationCurveLinear

If you set the same property for an object more than once, the first property change is animated, and then the last property change is animated when the first completes. Property changes between the first and last are ignored. The evAnimationsComplete event (for remote forms) is generated after the last property change(s) have completed. This allows you to reverse the effect of an animation (which is the equivalent to the autoreverse/repeat options available on iOS).

The About form in the example app is loaded into a subform using animations, which initially has the $alpha value of zero (fully transparent) and is increased to 255 during the animation, as follows:

```
# method behind About button
On evClick
  Switch iScreensize
    # set aboutSubForm size for different devices
    # etc.
  End Switch
  Do $cinst.$objs.aboutSubForm.$classname.$assign("jsAbout")
  Do $cinst.$objs.aboutSubForm.$visible.$assign(kTrue)
  Do $cinst.$beginanimations(500,kJSAnimationCurveEaseIn)
  Do $cinst.$objs.aboutSubForm.$alpha.$assign(255)
  Do $cinst.$commitanimations()
```

Time Zones and Dates

The JavaScript Client exchanges dates and times between the server and client using UTC time, regardless of where your server is located (note that UTC is essentially the same as GMT but UTC is used globally as the standard time for web servers). You should therefore store all dates and times in UTC and use the time zone offset of the client to either determine or set the local time of the client. The $construct row variable parameter for the remote task/form has a column JStimezoneOffset, which is the timezone offset in minutes for the client relative to UTC time. For example, if the client's local time zone is UTC+2 (or GMT+2), JStimezoneOffset will be 120. See the Construct Row Variable section for more information about the $construct row variable for tasks/forms.

Time Zone Functions

There are a number of Omnis functions that allow you to convert local dates and times to UTC since the client and server need to both use UTC time: they are listed in the 'Date and Time' group in the Omnis Catalog (F9/Cmnd-9).

- **loctoutc()** and **utctoloc()**
  Converts the specified *datetime* (or time) from the local timezone to UTC (Coordinated Universal Time), or vice versa, and returns the result
- **tzcurrent(), tzdaylight(), tzstandard()**
  Returns the character string identifying: the current time zone, the daylight saving time zone, or the standard time zone, respectively, of the current date and time of Omnis (plus tzoffset() returns the system time zone offset from UTC time in minutes)

You should note that on 32-bit Windows the TZ codes returned by the timezone (tz) functions are the long time zone names and not the short abbreviated time zone names. If you want to use the short time zone names, you can add a mapping to studio.stb, from the full name to the abbreviation you wish to use.

Local Time

The $localtime task property allows you to switch to Local time rather than UTC. If true, the JavaScript Client and the Omnis App Server exchange date-time values in local time rather than UTC time. $localtime must be set in design mode: it cannot be assigned at runtime.

Date and Time Conversion in SQLite

When converting Date and Time data in a SQLite database to dates in a remote form (i.e. in the JavaScript client), the subtype of any Date/Time data is taken into consideration. In this case, 'Short Date…' data subtypes will return a date only (no time component), and 'Time' data subtypes will return a time only (no date component).

PDF Printing

The **PDF Device** is a printing device (external component) to allow you to print a report from Omnis to a PDF file and display it in the JavaScript Client in the end-user's web or mobile browser. There are two client commands (used with $clientcommand) to allow you handle PDF reports (if the end user's device supports PDF). See the Report Programming chapter in the *Omnis Programming* manual for more information about creating report classes to format your PDF reports; this section describes the PDF Device.

There is an example app called **JS PDF Device** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

Supporting Files

The PDF Device component is available for Windows and macOS and is located in the 'xcomp' folder and is loaded automatically.

Node.js

Omnis uses **Node.js** to print reports which is installed ready to use in Omnis Studio. Note that Node.js is used for PDF printing in Omnis (as well as several other functions) subject to the MIT license from PDFKit: https://github.com/foliojs/pdfkit/blob/master/LICENSE

Note: In versions prior to Studio 11, Python (reportlab) was used for PDF printing, but this is no longer used and the python.zip file has been removed from the Omnis development tree. If you are converting to Studio 11, you do not need to adjust your application code or interface to use Node.js.

Fonts

The PDF device will only work with Reports that use **TrueType fonts,** and using other fonts may cause an error during PDF generation. Specifically only fonts contained in ".ttf", ".ttc" or ".dfont" files can be used. Therefore you must choose TrueType fonts for your report if you intend to print using the new PDF device.

PDF Print Destination

The component is called OmnisPDF and the "Print to PDF" option appears in the Print Destination dialog, available to end users from the main File menu. If the end user selects PDF as the report destination, then when they print a report, it will either be sent to a report file specified in $prefs.$reportfile, or if this preference is empty Omnis will prompt the user to select the path of the output PDF file. Note that this mechanism can be overridden programmatically by using the $settemp method (see below).

PDF Folder

You can specify an alternative folder in which to place PDFs, rather than using the default "omnispdf" folder. There is an item called "omnispdfFolder" in the "pdf" section in config.json that allows you to specify the path of a folder to receive PDFs, overriding the default location. The item defaults to empty, which means Omnis will use the current "omnispdf" folder. The folder specified in "omnispdfFolder" must already exist, otherwise Omnis reverts to the default omnispdf folder.

Printing PDF Using Code

To send a report to PDF programmatically, you can use the following code:

```
Calculate $cdevice as kDevOmnisPDF
Set report name MyReportClass
Print report
; or
Calculate $cdevice as kDevOmnisPDF
Set reference lReportInst to $clib.$reports.NewReport.$open('*')
Do lReportInst.$printrecord()
Do lReportInst.$endprint()
```

In both of these cases the report will be sent to a report file specified in the $prefs.$reportfile preference. Note that there is a separate value of $prefs.$reportfile for each Omnis App Server stack (thread) and the main Omnis thread. If $prefs.$reportfile is empty, and the code is running in the main Omnis thread, Omnis will prompt the user for the destination PDF file; otherwise, if $prefs.$reportfile is empty and the code is running in another thread, Omnis will generate a runtime error.

PDF Device Functions

The PDF device has a number of functions to allow you to set up reports sent to temporary PDF files, to set up document properties, and to add security features such as passwords and encryption.

$settemp()

The $settemp function allows you to specify that the next report will print to a temporary PDF file in the omnispdf/temp folder. The function returns the name of the file that will be created in omnispdf/temp (or an empty string if bTemp is kFalse). You can specify a timeout in minutes whereupon the temporary PDF file will be deleted.

```
Do Omnis PDF Device.$settemp(bTemp,iTimeout) Returns cID
```

Parameter	Description
bTemp	A Boolean: kFalse means a temporary file will not be used. kTrue means the next report sent to PDF by the current task instance will be written to a temporary file in the folder “omnispdf/temp” in the data part of the Omnis Studio tree. Note that after the next report has been sent to PDF, the stored value of bTemp will revert to kFalse.
iTimeout	An integer: Only used when bTemp is kTrue. If omitted defaults to 10. The time in minutes for which the next PDF file generated by the current task will remain on disk. When the time expires, the Omnis PDF device automatically deletes the file. Note that Omnis automatically deletes any files left behind in omnispdf/temp when it starts up.

Each task instance (including remote tasks) stores its own information set up using $settemp. This allows $settemp to be used in one thread on the Omnis Server without effecting other threads/clients.

$setdocinfo()

The $setdocinfo function lets you specify the author, title and subject properties for the PDF documents generated by the current task. The author, title and subject parameters are all strings, and the function returns kTrue for success.

```
Do Omnis PDF Device.$setdocinfo(cAuthor,cTitle,cSubject) Returns bOK
```

You can add keywords to the PDF file’s metadata by adding extra parameters to $setdocinfo, specified as a string of comma-separated keywords, for example:

```
Do Omnis PDF Device.$setdocinfo(cAuthor,cTitle,cSubject,'keyword1, keyword2, keyword3') Returns bOK
```

$encrypt()

The $encrypt function sets encryption (security) properties for the PDF documents generated by the current task, and the function returns kTrue for success. The full syntax is:

```
Do Omnis PDF Device.$encrypt(cUserPassword[,cOwnerPassword='',bCanPrint=kTrue,bCanModify=kFalse,bCanCopy=kTrue
```

The parameters are as follows:

Parameter	Description
cUserPassword	A character string: The user password for the document. If this is set to empty then none of the other arguments apply and the document will not be encrypted; otherwise the document will be encrypted and the user password and other properties specified by this function will be applied to it.
cOwnerPassword	A character string: The owner password for the document. The default is no password is assigned.
bCanPrint	A Boolean: Specifies if the user can print the PDF document. The default is kTrue.
bCanModify	A Boolean: Specifies if the user can modify the PDF document. The default is kFalse.
bCanCopy	A Boolean: Specifies if the user can copy from the PDF document. The default is kTrue.
bCanAnnotate	A Boolean: Specifies if the user can annotate the PDF document. The default is kFalse.

Security in Third-party PDF readers

The optional security parameters will be applied to the PDF file if you include them in the $encrypt() function, but you should note that the third-party PDF viewer the end user is using may not support these settings or may just choose to ignore them. The password specified in cUserPassword should be interpreted by all PDF readers.

Completing PDF Printing

When a PDF report completes, the device sends a message to the task (or remote task) that printed the report. The message is $pdfcomplete, and it takes two arguments:

1. The pathname of the output report file.

2. A Boolean which is true for success, false if an error occurred generating the PDF.

Icons

You can include icons in text in a report printed to PDF using the style() function and the kEscBmp escape constant. For example, you can use con(style(kEscBmp,1400),’some text’) in a report entry field calculation to display an icon on a report.

PDF Printing in the JavaScript Client

PDFs generated by the PDF device can be used with the JavaScript Client. The **showpdf** and **assignpdf** client commands can be used with $clientcommand to display PDF files on the client. You should avoid generating large PDF documents to use with the JavaScript Client since the generated PDFs are streamed from the Omnis App Server (i.e. via a Web Server). An alternative would be to output the PDF document into the Web Server’s file system and link to it using a URL to the PDF on the Web Server and the **getpdf** command to send it to the client: see Linking to PDFs.

There is an example app called **JS PDF Device** in the **Samples** section in the **Hub** in the Studio Browser showing how you can print reports to PDF in the JS Client.

showpdf

The **showpdf** client command opens the specified PDF in a new browser window or tab. There is no control over whether the PDF is opened in a new window or tab: typically, this depends on the end-user browser settings, including their setting for popups.

```
Do $cinst.$clientcommand("showpdf",rowVariable)
```

Where *rowVariable* is row(**pdf-id, timeout, pdf-filename**)

- **pdf-id**
  A character string. Either a full pathname of a PDF file on the Omnis server (pdf folder must be specified in getpdfFolders config.json item; see below), or an id returned by $settemp
- **timeout**
  An integer being the time in seconds that the client is prepared to wait for PDF generation to finish. Defaults to 60. If PDF generation does not complete in time, or an error occurs, Omnis returns a PDF document containing a suitable error message.
- **pdf-filename**
  the file name of the PDF file

The following method generates a PDF and displays it on the client:

```
Calculate $cdevice as kDevOmnisPDF
Do Omnis PDF Device.$settemp(kTrue,1) Returns lID
Set report name New Report
Do Omnis PDF Device.$encrypt('bob','owner',1,0,0,0)
Do Omnis PDF Device.$setdocinfo('Bob','Title','Subject')
Print report
Do $cinst.$clientcommand("showpdf",row(lID,20))
```

assignpdf

The **assignpdf** client command opens the specified PDF in a PDF viewer control in the current remote form instance. The PDF must be assigned to an HTML control in the remote form which tries to open the PDF file using the PDF viewer installed in the end user's browser.

```
Do $cinst.$clientcommand("assignpdf",rowVariable)
```

Where *rowVariable* is row(**html-object-name, pdf-parameters, pdf-id, timeout, pdf-filename**)

- **html-object-name**
  The name of an HTML Object control in the current remote form instance. The HTML content of this object will be replaced with that necessary to display the PDF document in a PDF viewer control.
- **pdf-parameters**
  PDF viewer parameters. These apply when the PDF is viewed in a browser that uses the standard Adobe PDF viewer control. They control the look and behavior of the PDF viewer. See the Adobe website for details about the PDF Open Parameters.
- **pdf-id**
  A character string. Either a full pathname of a PDF file on the Omnis server (pdf folder must be specified in getpdfFolders config.json item; see below), or an id returned by $settemp
- **timeout**
  An integer being the time in seconds that the client is prepared to wait for PDF generation to finish. Defaults to 60. If PDF generation does not complete in time, or an error occurs, Omnis returns a PDF document containing a suitable error message.
- **pdf-filename**
  the file name of the PDF file

The following method creates a report and assigns it to an HTML control in the remote form.

```
Calculate $cdevice as kDevOmnisPDF
Do Omnis PDF Device.$settemp(kTrue,1) Returns lID
Set report name New Report
Do Omnis PDF Device.$encrypt('bob','owner',1,0,0,0)
Do Omnis PDF Device.$setdocinfo('Bob Smith','Title','Subject')
Print report
Do $cinst.$clientcommand("assignpdf",row("htm","toolbar=1&zoom=20",lID,10))
```

If you set the $html property of the HTML Object control to "<div %e></div>" the PDF viewer will have the same border, position and dimensions as the designed control on the remote form. Note that this command will not work on Android with the default web browser, since it does not support the application/pdf plug-in. If the application/pdf plug-in is not available on Android, it executes showpdf instead.

PDF Path Names

The character length of the path name when creating a PDF is unlimited (in versions prior to Studio 11, the limit was 255 characters). Under Windows, for very long path names, you may need to enable long paths by setting the registry key Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\LongPathsEnable to 1 and restart.

PDF Version and Encryption

You can set the PDF version and encryption used in the PDF file using the **$setpdfversion** method, which can be one of the kDevOmnisPDFVersion… constants identifying the PDF version and encryption to be used when encrypting the PDF file. The default is kDevOmnisPDFVersion13 which specifies 40-bit RC4 encryption. The following constants are available:

Constant	Description
kDevOmnisPDFVersion13	Version 1.3, 40-bit RC4 encryption
kDevOmnisPDFVersion14	Version 1.4, 128-bit RC4 encryption
kDevOmnisPDFVersion15	Version 1.5, 128-bit RC4 encryption
kDevOmnisPDFVersion16	Version 1.6, 128-bit AES encryption
kDevOmnisPDFVersion17	Version 1.7, 128-bit AES encryption
kDevOmnisPDFVersion17ext3	Version 1.7 ExtensionLevel 3, 256-bit AES encryption

PDF/A support

You can set the PDF/A subset type for a PDF file, which is used to create archival versions of documents. The standard version PDF/A-1 is supported, as well as PDF/A-2 a/b and PDF/A-3 a/b.

You can use the **$setpdfsubset** method to set the PDF/A subset type using one of the following constants:

Constant	Description
kDevOmnisSubsetPDFA1a	PDF/A-1a - Part 1 Level A (accessible) conformance
kDevOmnisSubsetPDFA1b	PDF/A-1b - Part 1 Level B (basic) conformance
kDevOmnisSubsetPDFA2a	PDF/A-2a - Part 2 Level A (accessible) conformance
kDevOmnisSubsetPDFA2b	PDF/A-2b - Part 2 Level B (basic) conformance
kDevOmnisSubsetPDFA3a	PDF/A-3a - Part 3 Level A (accessible) conformance
kDevOmnisSubsetPDFA3b	PDF/A-3b - Part 3 Level B (basic) conformance
asof 35439 kDevOmnisSubsetNone	Unsets the PDF subset

If you wish to unset the PDF subset, you can use the kDevOmnisSubsetNone constant with the $setpdfsubset() method.

Linking to PDFs

You can use the **getpdf** parameter when executing a URL such as the following to reference a PDF:

```
http://127.0.0.1:5912/jsclient?__OmnisCmd=getpdf,C:\myreport.pdf
```

To enhance security, the getpdf parameter only gets files with a ".pdf" extension, and it only works with parameters that are associated with an open remote task instance which is stamped with a creation time value.

Setting PDF Folders

To enhance security, you can limit the folders from which PDF files can be retrieved using **showpdf, assignpdf,** or **getpdf** by specifying the folders in the config.json configuration file, and thereby excluding all other folders. To do this you need to add the **getpdfFolders** item to the "server" member in config.json and list the folders. For example:

```
 "getpdfFolders": [
    "c:\\dev\\unicoderun",
    "c:\\dev\\temp"
  ]
```

The item can be an array of folder paths, and any subfolders of a configured folder is also allowed. Each entry is a valid folder path, without a trailing file separator.

Print PDF Example

The following methods will allow the end user to print a report to a PDF file: these methods use the built-in PDF Device methods and the client commands.

```
# button to open PDF report
# create var lID (Char)
On evClick
  Do $cinst.$createReport() Returns lID
  Do $cinst.$clientcommand("showpdf",row(lID,60))
# or button to display PDF report in current remote form
# create var lID (Char), oHTML is an HTML obj on the form
On evClick
  Do $cinst.$createReport() Returns lID
  Do $cinst.$clientcommand(  "assignpdf",row("oHTML","toolbar=1&zoom=20",lID,20))

# the code for the $createReport() remote form method:
Calculate $cdevice as kDevOmnisPDF
If iSaveCopy ;; linked to check box on the form
  Do Omnis PDF Device.$settemp(kFalse) Returns lID
  Calculate lSaveLocation as left(sys(10),rpos(sys(9),sys(10)))
  Calculate lSaveLocation as
   con(lSaveLocation,"savedReports",sys(9),iFileName)
  Calculate $prefs.$reportfile as lSaveLocation
Else
  Do Omnis PDF Device.$settemp(kTrue,1) Returns lID
End If
Set report name repNice
If iEncrypt
  Do Omnis PDF Device.$encrypt(
    iUserPass,iOwnerPass,iCanPrint,iCanModify,
    iCanCopy,iCanAnnotate) Returns #F
End If
Do Omnis PDF Device.$setdocinfo(iAuthor,iTitle,iSubject) Returns #F
```

```
Print report
If iSaveCopy
  Quit method lSaveLocation
Else
  Quit method lID
End If
```

Toast Messages

Toast messages are small notification type messages that that can be "popped up" in a remote form to alert the end user about something: the concept is derived from "toast messages" on Android. (Note this section refers to toast messages for *remote forms* and not desktop toast messages.)

Toast messages are activated using the "showtoast" client command ($cinst.$clientcommand) which displays a message to the user in a small popup window which disappears after a timeout, either 5000ms or specified amount.

```
Do $cinst.$clientcommand("showtoast",rowVariable)
```

Where rowVariable is row(text, [timeout, posX, posY, containerName, floating, originX, originY, speakMessage, assertive])

The toast message rowVariable parameters are:

Parameter	Description
text	The message text. The container's size will scale with the amount of text. You can use '\n' to insert a new line.
timeout	(Optional) The length of time (ms) the message will be shown for (5000 ms by default).
posX	(Optional) The horizontal position of the toast message in pixels. Centered if not specified. If containerName is specified, this position is relative to the control.
posY	(Optional) The vertical position of the toast message in pixels. Positioned near the bottom of the form if not specified. If containerName is specified, this position is relative to the control.
containerName	(Optional) The name of the control to position the toast message relative to. Options are limited to paged pane and subform controls.
fixed	(Optional) This determines whether or not the toast message will stay in position when the container scrolls (true by default).
originX	(Optional) The toast message's origin that posX references. Possible values are: kLeftJst (default), kRightJst and kCenterJst.
originY	(Optional) The toast message's origin that posY references. Possible values are: kJstVertTop (default), kJstVertMiddle and kJstVertBottom.
speakMessage	(Optional) If true, screen readers will announce the message. This is an accessibility feature to convey information to visually impaired users.

Parameter	**Description**
assertive	(Optional) If speakMessage is true, this instructs screen readers whether or not to interrupt current speech.

The originX and originY parameters are used to set the point on the toast message that posX and posY reference. For example, if originX is kRightJst and originY is kJstVertBottom, the bottom right corner of the toast message will be at the position specified by posX and posY.

Chapter 3—JavaScript Components

The **Component Store** contains over 40 ready-made components for use in Remote Forms and the JavaScript Client. The components are arranged in functional groups in the Component Store, including a group for Buttons, Containers, Entry Fields, and so on. The JavaScript components are listed in this chapter in alphabetical order, starting with the Activity Control, and are listed in the table below in their respective groups.

You can create your own JavaScript components using JavaScript and C++: see the JavaScript Component SDK online doc which has a tutorial and component reference. Alternatively, you can create your own custom JavaScript components, defined using JSON, which will appear in the **JSON Components** group in the Component Store: these are described in the JSON Components chapter.

Example Apps and Code

There is an example app for most of the JavaScript components under the **Samples** option in the **Hub** in the Studio Browser. Open these apps to examine the remote forms and look at the code behind each component: you can double-click on a JavaScript component in design mode to see its code methods in the method editor.

In addition, the **Applets** option in the **Hub** in the Studio Browser has several sample web apps that use many of the JavaScript components, including a **Contacts** app, a **Holidays** app, and a **Webshop** app which has a product catalog and a prototype shopping cart.

The same example apps are featured in the **JavaScript Apps Gallery** on the Omnis website, available here: www.omnis.net/platform/#jsgallery

or you can use the following shortcut to open the **Omnis Components** gallery: tinyurl.com/jsgallery11

JavaScript Components

The following components are available in the Component Store, arranged here in their respective groups.

Group	Icon	Name	Description
Favorites	★	See Favorites	Contains any controls marked as F
Buttons	OK	Button Control	Standard pushbutton which reacts
	☑ Opt1 ☑ Opt2	Check Box Control	Check box for on/off values
	+	Floating Action Button	A round button that pops up a list actions when tapped or hovered o
	◉ Opt1 ◉ Opt2	Radio Button Group	Displays a group of radio buttons f exclusive selection

Group	Icon	Name	Description
	OK	Split Button	A button with a droplist of alternat options
		Switch Control	Allows on/off selection; you can spe icon for on/off state
		Trans Button Control	Interactive button with alternate h image
Containers		Paged Pane	Can contain fields & other objects o multiple panes
		Scroll Box	Allows you to group other controls option to display a scroll bar if the c does not fit
	Tab1 TAB2	Tab Pane	A compound object containing a T and Paged pane
Entry Fields	Ab	Entry Field	Standard edit field for data entry o
	B I U • Bold • Italic	Rich Text Editor	Rich text editor allowing end users and format text
Labels	T	Label Object	Basic label object
Lists		Combo Box Control	Field combining entry box and dro
		Complex Grid	Grid which can display all types of formatting
		Data Grid Control	Simple grid for text and numerical display
		Droplist Control	List that drops down when clicked
		List Control	Standard list field for displaying list data
		Tile Grid	Displays a scrollable grid of tiles wh be configured to show images, tex buttons

Group	Icon	Name	Description
		Tree List Control	List for displaying hierarchical data options
Media		Camera Control	Allows the end user to capture ima scan QR codes or barcodes
		File Control	Allows end users to upload or dow files
		HTML Object	Object to display HTML content
		Picture Control	Standard field for displaying image
		Video Player	Plays a YouTube or other hosted vi
Menus		Popup Menu Control	A menu that pops up when clicked
Native		Native List	List control with platform depende appearance
		Native Slider	Slider control with platform depen appearance
		Native Switch	Switch control with platform depe appearance
Navigation	LINK LINK LINK	Hyperlink Control	List containing hyperlink style opti
		Navigation Bar Control	Navigation bar with page selection
		Navigation Menu Object	Dropdown menu with hierarchical
	◆◆◆	Page Selector	Allows selection of page pane usin
		Segmented Bar	Navigation control with different b “segments”
	Tab1 Tab2	Tab Bar Control	Multiple tabs to control selection o pane

Group	Icon	Name	Description
Other		Toolbar Control	Toolbar with custom buttons (icon text), auto overflow and optional si
		Activity Control	Animated image to display during process or Omnis Server activity
		Color Picker	Allows the end user to select a colo color palette, or RGB, HSL, or HEX r
		Date Picker Control	Date picker with touch selection
		Device Control	Allows access to hardware and ser a mobile device using the JS wrap
		Map Control	Displays a Google map for specifie location(s)
		Progress Bar Control	Shows progress of server process c calculation
		Slider Control	Slider component for setting value
		Timer Control	Timer object triggers an event at a specified interval
Shapes		Background Shape	Object you can set to Rectangle, Li Triangle, or Image
Subforms		Subform Control	Allows you to display another remo class as a subform in the main form can create a subform set)
Visualization		Bar Chart Control	Displays a bar chart based on a list
		Chart Control	Displays different chart types inclu Line, Bar, Radar, Pie, Doughnut, Po Scatter and Bubble
		Gauge Control	Displays numerical values on a circ linear scale

Group	Icon	Name	Description
	◔	Pie Chart Control	Displays a pie chart based on a list

Favorites

The **Favorites** group contains any components that you have marked as 'Favorite'; it is shown initially with a Star icon and grayed out. To add a favorite, Right-click on *the icon for the component* in a sub-menu and select the **Favorite** option. Adding components to the Favorites group makes it easier or quicker for you to select any controls that you use constantly. To remove a component from the Favorites group, right-click on the component in its original group and *deselect* the **Favorite** option.

Creating JavaScript Components

To create or add a JavaScript Component to a remote form, you need to open the Remote form in design mode. The Component Store will open automatically docked to the left side of the Remote form editor. If for some reason the Component Store is hidden (maybe it is undocked and behind another window), you can bring it to the top, by pressing the **CStore** button on the main Omnis toolbar, or by pressing the **F3** key on Windows or **Cmnd-3** on macOS. The following screenshot shows a Remote form in design mode, and the Entry fields group in the Component Store.

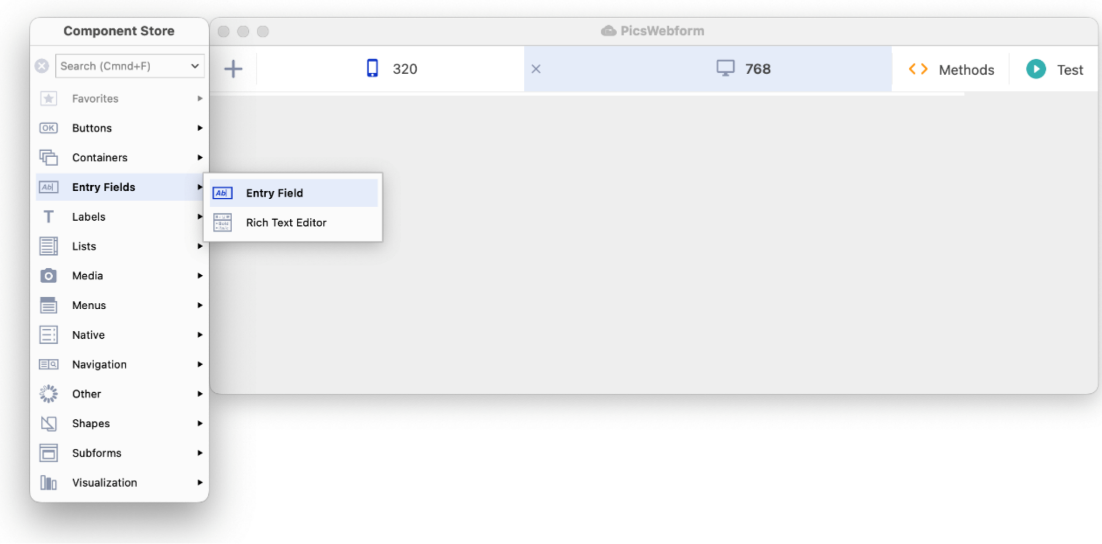


Figure 101:

You can configure the appearance of the Component Store by Right-clicking on it and selecting various text options from the context menu. You can show or hide the text for the group or components, and you can select 1 or 2 column mode. You can also set the Docking mode, either Auto, Left, Right, or No docking.

Layout breakpoints

Before adding any JavaScript components to your remote form, you may want to change the current *Layout Breakpoint* or add a new one. A new remote form has two breakpoints **320** and **768** pixels, and you are advised to add components to the larger breakpoint first and then rearrange or resize the components on the smaller breakpoint (the 768 breakpoint is selected by default). You can Right-click on the background of a remote form and choose **Copy Layout From Breakpoint** to copy the position and size of components from one breakpoint to the current breakpoint.

Adding a new JavaScript component

To select a component, and add it to your Remote Form, you can do one of the following:

- *Click on the main group icon* to open the sub-menu popup, then *click and drag a component icon* from the sub-menu, and drop it onto the form or window; as you drag the component out of the Component Store, the outline of the component is shown allowing you to place it precisely in the form or window.
- *Click and drag the icon shown in the main group* to create a component of that type; for example, you can drag the Button icon from the Buttons group to create a button, which is initially the default icon in that group (note the group icon/default component will change as you select different components).
- *Double-click an icon* in the main group or any sub-menu popup to add a component of that type; in this case, the component is *added to the center* of the form or window (double-clicking is not supported for report classes).
- *Press Return* to add the *currently selected component* to the design window (not supported for report classes).

Alternatively, you can use the keyboard to select a component:

- *To use the keyboard,* press **F3** to put the focus on the Component Store, use the **Up** or **Down** arrow keys to select a main group, press the ***Space*** key to open the sub-menu popup for the group, then *use the Arrow keys* to select a component, *and* press the ***Return*** key to add the component to the center of the form or window; you can use the ***Esc*** key to deselect/close a sub-menu popup.

The most recently selected group is highlighted in a color, while the icon for the most recently chosen component from any sub-menu popup is shown as the initial/default icon for the group; therefore, as you select different components from different groups, the initial or default icons will change. For example, if you previously chose a Combo box from the Lists group, the Combo box icon is shown in the main Lists group, and you can then drag or double-click the Combo box icon from the Lists group without opening the sub-menu to create a Combo box in your form.

Remote Form Object Limit

You cannot place an unlimited number of objects on a Remote form class. The object limit is **8191** for a Remote form, including objects on subforms, although in practice the limit is likely to be less due to platform limitations.

Copying Components

In design mode, you can use the standard Copy/Paste menu options in the **Edit** menu, or Ctrl-C and Ctrl-V keyboard options, to copy and paste a component on the same form. Alternatively, you can hold down the **Ctrl** key on Windows or the **Alt/Option** key on macOS, then *click and drag* a component to a new position on the form to make a copy of the component.

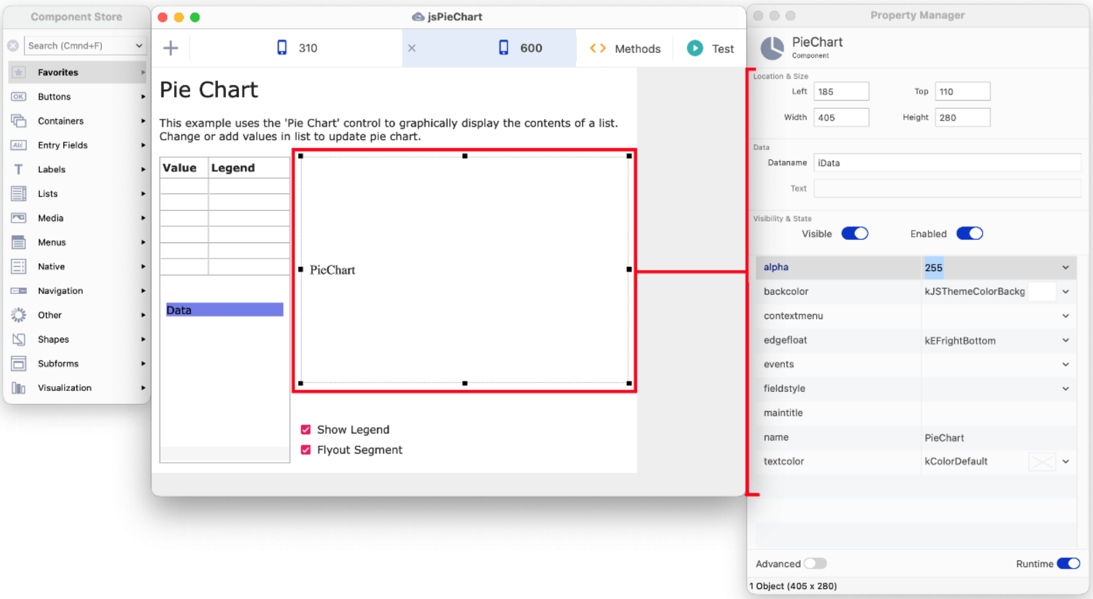
You can also drag a component from *one remote form and drop it onto another* remote form to make a copy of a component (for forms with the old kLayoutTypeScreen layout type, the forms must be *set to the same value of $screensize* to copy objects in this way).

Component Properties

You can set the properties for the component using the **Property Manager;** if the Property Manager is hidden, press **F6** on Windows or **Cmnd-6** on macOS to open it or bring it to the top. The following screen shot shows the Pie Chart example app (available in the Hub under Samples), with the JavaScript *Pie Chart* component selected; the *Property Manager* on the right shows the properties of the currently selected object.

$dataname for JavaScript Controls

The *variable* specified in the $dataname property of a JavaScript component *must be an instance variable,* or in some cases a *column* in a *row instance variable* in the form VarName.ColumnName. The Property Manager will display an error message if you try to assign an invalid $dataname property. (This applies to $dataname as well as other similar properties such as $listname which require a variable.)



Pie Chart

This example uses the 'Pie Chart' control to graphically display the contents of a list. Change or add values in list to update pie chart.

Value	Legend

Data

PieChart

☑ Show Legend
☑ Flyout Segment

Location & Size
Left 185 Top 110
Width 405 Height 280

Data
Dataname iData
Text

Visibility & State
Visible ● Enabled ●

alpha	255
backcolor	kJSThemeColorBackg
contextmenu	
edgefloat	kEFrightBottom
events	
fieldstyle	
maintitle	
name	PieChart
textcolor	kColorDefault

Advanced ○ Runtime ●

1 Object (405 x 280)

Figure 102:

Naming JavaScript Controls

When you create a component in your remote form, a name is generated automatically and assigned to the $name property of the component. This is usually in the format <remoteformname>_<component-type>_<number>, such as 'rftest_edit_1001' for an edit control on a remote form called rftest. However, you can enter your own name for a component which may better describe the object within the context of your form; for example, an edit field to allow the end user to enter their first name could be named *Firstname*. You can change $name of a component under the General tab in the Property Manager, or enter name into the Search to find the $name property.

The name you assign to an object does not have to conform to any convention other than any conventions you may like to use in your forms or the application to identify different objects. However, the name of a component (the value of $name) is used in the Omnis notation and throughout your library to refer to the object. Therefore, you should ***not use spaces*** and try to use ***alphanumeric characters only*** for object names to avoid any possible conflicts in your code. For example, an object name ***should not include*** the dollar sign ($) since this would cause a conflict when you reference the object using the Omnis notation which prefixes property and method names with the dollar sign.

Numeric Object Names

The Property Manager does not allow all numeric names to be assigned to $name. The Property Manager validates the value assigned to $name for remote form objects (as well as remote menu, report, schema, menu, toolbar and window class objects). The validation is applied when the name starts with a digit, and the remaining characters cannot all be a digit or the following characters "+-.".

This is controlled by the **allowNumericObjectNames** item in the 'ide' section of config.json. You are not recommended to allow numeric object names, as there can be clashes between names and idents, and notation strings of the form …$objs.[lName] (where lName is a variable containing the name of an object) will fail to locate the object if lName is an integer, since Omnis will treat lName as an ident rather than a name.

Using $edgefloat and Component Resizing

The "floating edge" ($edgefloat) capabilities for JavaScript components allow the components to be resized automatically when the end user resizes their web browser window or when the layout changes on a breakpoint. The $edgefloat property can be set to one of the kEF… constants which determines which edges of the component, if any, will "float" or reposition automatically when the browser window is resized. The possible values for $edgefloat are:

- **kEFall** and **kEFnone**
  All or no edges float

- **kEFbottom**
  Bottom edge only floats

- **kEFleftRight**
  Left and right edges float; in effect, the component floats to the right or left and does not resize

- **kEFleftRightBottom**
  Left, right and bottom edges float

- **kEFright** and **kEFrightBottom**
  Right edge only floats, or Right and bottom edges float

- **kEFtopBottom**
  Top and bottom edges float; in effect, the component floats up or down and does not resize

- **kEFrightTopBottom**
  Right, top and bottom edges float

- **kEFcenterLeftRight, kEFcenterTopBottom, kEFcenterAll**
  means the Left & Right edges float, or the Top & Bottom edges float, or All edges will float, *and* the control will also be centered horizontally and/or vertically within its parent

- **kEFbottomAndCenterLeftRight**
  the *bottom edge* of the object will float or move up or down, while the object stays centered horizontally in the form (a combination of kEFbottom and kEFcenterLeftRight)

- **kEFrightAndCenterTopBottom**
  the *right edge* of the object will float or move to the right or left, while the object stays centered vertically in the form (a combination of kEFright and kEFcenterTopBottom)

- **kEFleftRightAndCenterTopBottom**
  the control floats with the right edge of its container, and remains centered vertically (a combination of kEFleftRight and kEFcenterTopBottom)

- **kEFtopBottomAndCenterLeftRight**
  the control floats with the container's bottom edge, and remains centered horizontally (a combination of kEFtopBottom and kEFcenterLeftRight)

- **kEFposn… positioning constants**
  all edgefloat constants prefixed with kEFposn… will reposition the control in the specified region of the screen; as you select one of these constants in design mode the control will snap to the chosen region, and when the form is resized at runtime the control will "stick" to this region; the **kEFposnClient** constant stretches the control to fit the available area within its parent or subform

You can store a different setting of the $edgefloat property for each component, for each different layout breakpoint. When setting $edgefloat in the Property Manager in design mode, you can set the value of $edgefloat for a component on all breakpoints by holding the Control key when selecting the $edgefloat value.

The setting of $edgefloat for a component is used to resize the component (or not if set to kEFnone) when the form or container field is resized at runtime, and when one or more of the following occurs:

- When the component is in a subform and the subform is resized (that is, its size at runtime is different to the size of the subform class)

- When applying a different mobile device size while running in a mobile device custom wrapper

- When the component is in a resizable subform in a subform set and the subform is resized

Centering Objects

There are some kEF… contstants to control how objects are centered relative to the remote form or parent: kEFruntimeLeftRightCenter, kEFruntimeTopBottomCenter and kEFruntimeAllCenter. They are only applied at runtime, and in this case, their behavior is identical to kEFleftRight, kEFtopBottom or kEFall respectively, except that the offset is divided by two, to keep an object or a number of objects centered within the parent.

In addition, the Align context menu for the remote form editor contains options to allow you to center objects vertically, horizontally (or both) in their parent.

Responsive Forms and $edgefloat

To understand what kind of edgefloat properties you can use, you can look at the **PicsWebForm** in the Pics2.lbs available in the tutorial download (or in the 'welcome/tutorial/final' folder); or you could create your own remote form using the form wizard. The PicsWebForm remote form was created using the SQL Remote Form wizard and uses edgefloat properties to control the floating edge behavior of the controls. The form has two layout breakpoints, 768 and 320, and the edgefloat properties is set differently for some of the controls on each breakpoint. The following image shows the layout for the 768 breakpoint, and the $edgefloat setting (a kEF.. constant) for each control is shown in red.

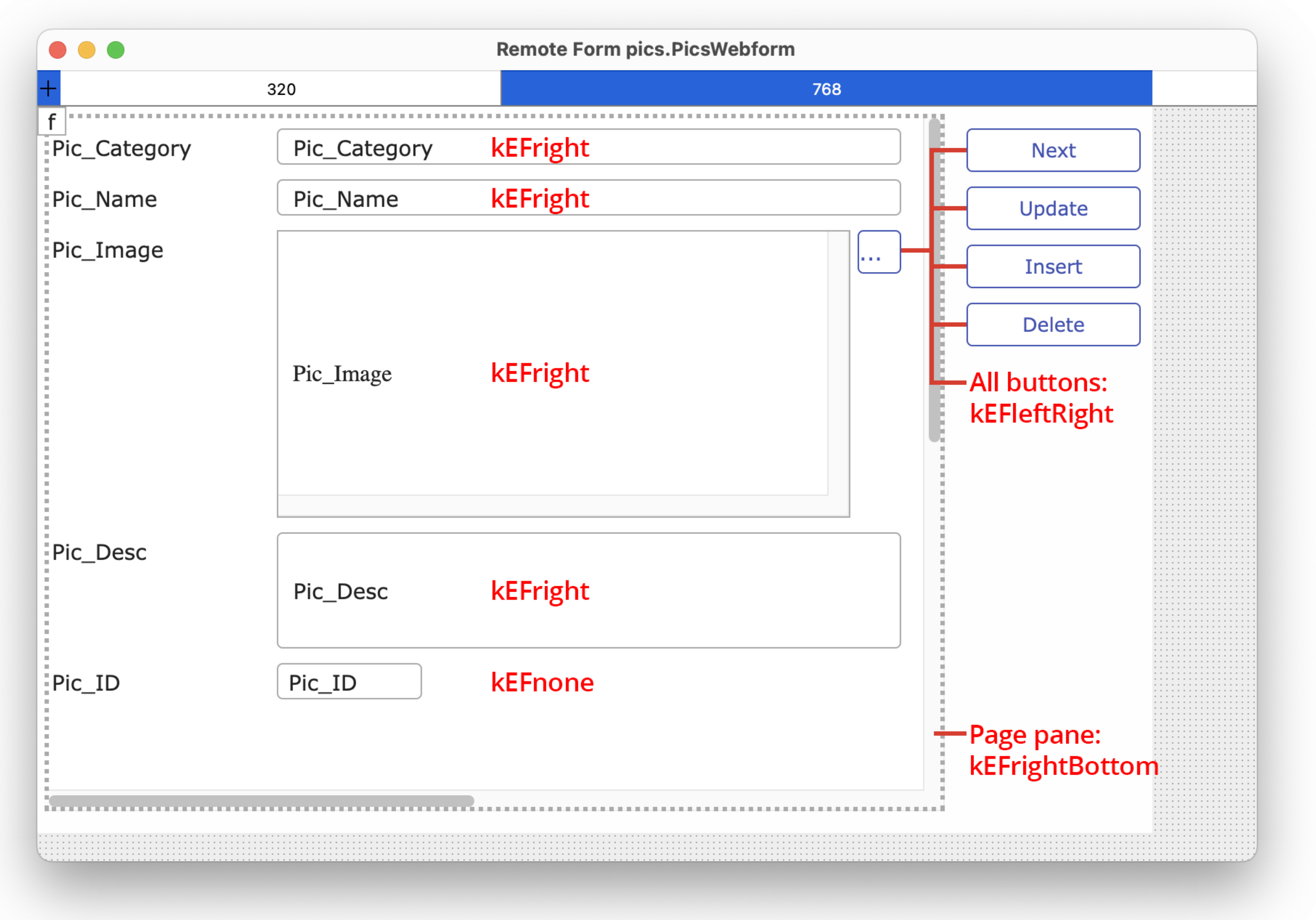


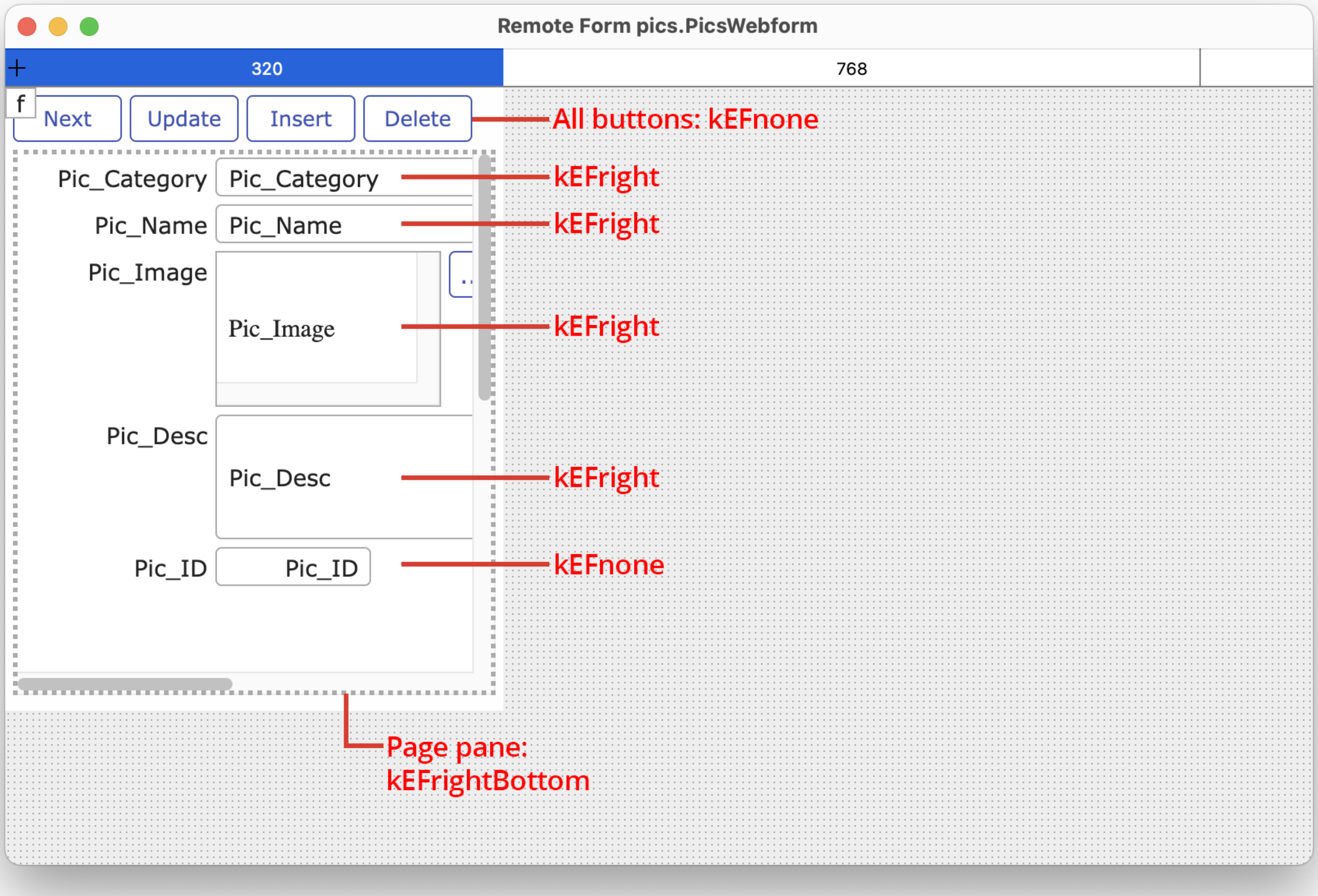
Figure 103:

The PicsWebForm uses a Page pane containing all the data controls, e.g. Pic_Category, Pic_Name, etc; the $edgefloat property of the Page pane is set to kEFrightBottom to ensure it stretches across to the right and down as the form is resized in a browser window or is displayed on different sized tablet screens.

The $edgefloat property for most of the controls inside the page pane is set to kEFright, so the right edge "floats" or stretches to the right, but the bottom edge is not resized; the ID field has no floating edges so it keeps its size. The $edgefloat property for the push buttons on the right of the form is set to kEFleftRight (i.e. both left and right edges), which means the buttons will "float" from right to left horizontally, but they will not resize or move vertically. The combination of all these edgefloat settings on all of the controls, means that the push buttons keep to the right-hand edge of the browser window or device screen, while the data controls will resize to accommodate any screen or device size. Now examine the layout for the 320 breakpoint:

The push buttons on the 320 layout breakpoint are positioned at the top of the form and their $edgefloat property is set to kEFnone, so they will not move or resize as the form is resized. The $edgefloat property for the data controls is set to kEFright so their right edges will stretch to accommodate different phone sizes (widths), from 320 pixels upwards.

As the form is resized, on a web browser window or is displayed on a larger device screen, the controls will resize to fill the screen, until the next breakpoint is reached, which in this case is a screen or device width of 768 pixels, and the layout for that breakpoint is loaded.



All buttons: kEFnone

kEFright

kEFright

kEFright

kEFright

kEFnone

Page pane:
kEFrightBottom

Figure 104:

Draggable Component Borders

End users can resize some JavaScript components dynamically at runtime in their web browser by dragging the border of the component. When the end user's mouse is over the edge of a component that can be resized, the cursor changes to indicate that the border can be dragged and resized.

To allow this functionality, JavaScript components have the $dragborder property, which only applies when a component has its $edgefloat property set to one of the kEFposn... constants (other than kEFposnClient or kEFposnJoinHeaders). If $dragborder is set to true, and you have set $edgefloat as above, the end user will be able to resize the component in the browser by dragging the border of the component with the mouse.

You can store a different setting of the $dragborder property for each component, for each different layout breakpoint, therefore components on the same form could be resizable for web desktop browsers and not for mobile devices. When setting $dragborder in the Property Manager in design mode, you can set the value of $dragborder for all layout breakpoint values by holding the Control key when selecting the $dragborder value.

The appearance of the drag border area is defined by the styles div.omnis-db-vert and div.omnis-db-horz in core.css, which can be modified by overriding them in user.css.

Date and Time Formatting

You can set the formatting for Date and Time type data for some of the JavaScript components including Edit controls, Combo boxes, Data grids, Droplists, Hyperlink lists and standard Lists. These components have the properties:

- **$dateformatcustom**
  a date-time format string using the characters described below (e.g. D m y, the default); alternative formats can be provided separated by |. If $dateformat is kFormatCustom, and the data is of type 'Date Time', this property is used to format the data. If empty, it defaults to the format set using the client command 'setcustomformat'
- **$dateformat**
  the format used to display 'Date Time' data, a kJSFormat... constant as follows:

15% Format constant	85% Description
kJSFormatNone	No format
kJSFormatTime	Default time format for client locale
kJSFormatShortDate	Default short date format for client locale
kJSFormatShortDateTime	Default short date and time format for client locale
kJSFormatMediumDate	Default medium date format for client locale
kJSFormatMediumDateTime	Default medium date and time format for client locale
kJSFormatLongDate	Default long date format for client locale
kJSFormatLongDateTime	Default long date and time format for client locale
kJSFormatFullDate	Default full date format for client locale
kJSFormatFullDateTime	Default full date and time format for client locale
kJSFormatCustom	Use the custom format in $dateformatcustom

Date formatting characters

The following standard date formatting characters are supported for $dateformatcustom:

Format character	Description
A	AM/PM
D	day (12)
d	day (12th)
E	day of year (1..366)
H	hour (0..23)
h	hour (1..12)
M	month (06)
m	month (JUN)
N	Minutes

Format character	Description
n	month (June)
s	hundredths
S	seconds
V	day of week (Fri)
w	day of week (Friday)
y	year (1989)
Y	year (89)

Some additional characters are supported for Date/Time formatting for the JavaScript Client components only, as follows:

Format character	Description
j	day with no leading zero (6)
P	month with no leading zero (6)
K	hour with no leading zero (0..23)
k	hour with leading zero (1..12)
a	am/pm
O	timezone offset (+01:00)

The date codes are listed on the Constants tab in the Catalog (F9) under “Date codes” (some are not JS client) and “Date codes (JavaScript Client only)”.

Date formatting and Locale

When the client connects, the server sends it the date formats, day names and month names for the client locale (the server reads these from ICU). If you assign $ctask.$stringtablelocale in $construct of your remote task, the server sends the client the formats and so on for the assigned $stringtablelocale locale.

Date Initialization

Local Date variables in client methods with no initial value set are initialized to an empty string, i.e. representing an empty date (from Studio 10.22 onwards), whereas previously they were initialized to ‘undefined’. Setting a Date variable to 0 on the client now sets the date to 31 Dec 1900, whereas previously it was set to today’s date. To set a date to today’s date, you should use #D.

Number Formatting

All JavaScript controls that can display *number data* have the property $numberformat, which specifies how **Number** and **Integer** data is formatted or displayed in the control. The JavaScript controls affected include the Edit Control, Combo box, Data grid, Droplist, Hyperlink list and standard List control. The formatting is used when the control displaying the data does not have the focus, that is, the formatting is only applied when the end user tabs or clicks away from the number field.

The $numberformat property uses a single % format tag for the number followed by one or more elements, for example, the number format %.2F displays a number with 2 decimal places with a thousand separator. The following elements are available (in this order):

An optional “+” sign that forces to precede the result with a plus or minus sign on numeric values. By default, only the “-” sign is used on negative numbers.

An optional padding specifier used for padding (if padding is required). Possible values are 0 or any other character preceded by a ’. The default is to pad with spaces.

An optional “-” sign, that causes the string to left-align the result of this placeholder. The default is to right-align the result.

An optional number that says how many characters the result should have. If the value to be returned is shorter than this number, the result will be padded.

An optional precision modifier consisting of a “.” (dot) followed by a number, specifies how many digits should be displayed for floating point numbers. When used on a string, it causes the result to be truncated.

A type specifier that can be any of the following:

Type specifier	Description
	display a literal ” character
b	display an integer as a binary number
c	display an integer as the character with that ASCII value
d	display an integer as a signed decimal number
D	as above but include thousand separators
e	display a float as scientific notation
u	display an integer as an unsigned decimal number
f	display a float as is
F	as above but include thousand separators
o	display an integer as an octal number
s	display character(s) after the s specifier as a string, e.g. to display hours you could use sh to display 5h
x	display an integer as a hexadecimal number (lower-case)
X	display an integer as a hexadecimal number (upper-case)

Decimal and Thousand Separators

Numbers will be displayed using the default decimal and thousand separators specified by the language set in the client's browser, so you do not need to do anything to display the correct decimal and thousand separators for a client. However, you can override the default separators by changing the thouChar and dpChar items in the jOmnis client object: you can do this using JavaScript in the $init method for a JavaScript form, for example:

```
# $init method, which must be client executed
JavaScript:jOmnis.thouChar = ".";
JavaScript:jOmnis.dpChar = ",";
```

Autoscrolling

You can enable automatic scrolling for Edit controls, Lists, Tree lists, Hyperlink controls, Pictures and Html controls by enabling the $autoscroll property. If this property is kTrue for the control, and the client is not a mobile device, the client automatically displays scrollbar(s) when not all of the content in a field is visible.

Setting $autoscroll to kTrue changes $horzscroll and $vertscroll to kFalse, and in doing so means you cannot set $horzscroll and $vertscroll. By default, $autoscroll is enabled for Edit controls, Lists and Tree lists, while for Hyperlink controls, Pictures and HTML controls $autoscroll is set to kFalse.

Note that in addition to controlling scroll bars, Data Grids and Lists have the $vscroll and $hscroll properties which allow you to scroll a grid or list vertically or horizontally at runtime in the client browser: the numeric value of these properties is either column or row offset for grids, or the row number for lists.

Component Borders

The borders of JavaScript components are drawn *within* the bounds of the control, for both Windows and macOS, and have the same dimensions for both platforms. The color is controlled using $bordercolor.

Most JavaScript components can have rounded borders by specifying the corner radius in pixels in the $borderradius property; for buttons this is $buttonborderradius (single value only). To set all the corners of the object to the same radius you can enter a single value, or to specify the radius for different corners you can use the syntax “n-n-n-n” which follows the same rules as CSS 3 rounded border syntax. The order for the radius parameters is top-left, top-right, bottom-right, bottom-left. If bottom-left is omitted the top-right value is used, if bottom-right is omitted the top-left value is used, if top-right is omitted the top-left value is used.

Control Classnames

All JavaScript controls have a base class name to allow you to control the appearance of controls using CSS, to allow you to apply a consistent appearance for each type of JavaScript control. (Note that from Studio 10.2 onwards you can use JS Themes to manage the colors for controls on a remote form.)

The classnames listed below can be added to the ‘user.css’ and CSS properties applied to the classname to control the appearance of each type of control. Note these classnames are contained in the JavaScript controls by default and if they are added to the user.css are applied to the control automatically, that is, these classnames *do not* need to be included in the $cssclassname property of a control to be applied (this property is used to apply your own custom style names, see below).

JavaScript Control	Class Name
‘Frame’ element for all controls	omnis-[control]-frame
Activity Control	omnis-activity
Background Control	omnis-background
BarChart Control	omnis-barchart
Button Control	omnis-button
Checkbox Control	omnis-checkbox
Combo Box Control	omnis-combo
Complex Grid	omnis-complexgrid
Date Picker Control	omnis-date
Data Grid Control	omnis-datagrid
Droplist Control	omnis-droplist
Edit Control	omnis-input
File Control	omnis-file
HTML Object	omnis-html
Hyperlink Control	omnis-hyper
Label Object	omnis-label
List Control	omnis-list
Map Control	omnis-map
Menu - used for context menus, popup menus and tab menus	omnis-menu
Native List Control	omnis-nativelist
Native Slider Control	omnis-nativeslider
Native Switch Control	omnis-nativeswitch
Navigation Bar Control	omnis-navbar

JavaScript Control	Class Name
Navigation Menu Object	omnis-navmenu
Page Control	omnis-pagectl
Paged Pane	omnis-pagedpane
Picture Control	omnis-picture
Popup Menu Control	omnis-popup
PieChart Control	omnis-piechart
Progress Bar Control	omnis-progress
RadioGroup Control	omnis-radio
Rich Text Editor Control	omnis-rich
Segmented Control	omnis-segmented
Slider Control	omnis-slider
Subform	omnis-subform
Switch Control	omnis-switch
Tab Control	omnis-tabs
TransButton Control	omnis-trans
Video Control	omnis-video

For example, to add CSS styling to all the Edit controls in your remote forms you could add the following CSS to the user.css file in the 'html/css' folder in the main Omnis folder: in this case, the base classname .omnis-input is used with the properties 2px solid grey border and a 6px radius.

```
.omnis-input {
  border: 2px solid grey;
  border-radius: 6px;
}
```

Custom CSS Styles

You can create your own CSS classes or styles (in addition to the base class names listed above) and apply them to the objects in your web and mobile apps, allowing you to have more control of the styling, coloring, and overall design of your apps.

CSS classes for Controls

All the JavaScript components have a property called $cssclassname which allows you to apply your own CSS class to the component. You can add the CSS classes to a file called 'user.css' which is located in the 'html/css' folder in the main Omnis Studio folder. A style can be applied to a control by setting its $cssclassname property to the name of a style. The properties you define for each style in user.css must be flagged as !important to override the JavaScript Client inline styles.

$cssclassname adds the classes listed above to the client element in all JavaScript controls, and adds the same classes to the frame element with the "-frame" suffix.

When you deploy your application on the Omnis App Server, you must put your custom 'user.css' file in the 'html/css' folder on the server.

Component Transparency

The majority of the JavaScript components have the $alpha and $backalpha properties which allow you to set the transparency of the foreground and background colors of the component.

Form Errors

It is possible to display form errors either to the right or under controls in a remote form. This makes it easier for end users to fill out forms in your web and mobile applications by providing them with helpful tips if they make a mistake in the form. The errors appear on the form as a text field either under a control, or to the right of a control, so you need to design your remote forms to allow space for the error text. The style of the error text and the outer HTML of the control containing the error are controlled in CSS which you can change if required.

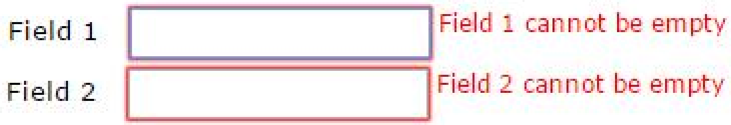


Figure 105:

$errortext

All JavaScript Client controls have the $errortext property which contains the text to be displayed when there is an error in the field or control. The initial value of $errortext when a form is constructed is empty.

The $errortext property is only supported for subform controls when they are not scrollable, i.e. when $vertscroll & $horzscroll are both kFalse and the subform class is not responsive.

$errortextpos

The $errortextpos property specifies the position of the error text set using $errortext. The value can be one of:

- **kJSErrorTextPosUnder**
  The error text is positioned under the control, the default.
- **kJSErrorTextPosRight**
  The error text is positioned to the right of the control.
- **kJSErrorTextPosHidden**
  hides the error text, so just the control outline indicates that there is an error (default is a red border). This might be useful where there is limited space to display the error text in the remote form, but you still want to show the end user that there was an error

Note that Omnis stores a separate value of $errortextpos for each screen size. As a shortcut when designing a form, you can hold the control key down when assigning $errortextpos, in order to assign the value to all screen sizes.

$errorline

$errorline is a runtime property of the JavaScript Complex Grid control, used when assigning $errortext to an object in the row section of a complex grid. The line number to which $errortext will apply.

Clearing form errors

The client command “clearerrors” allows you to clear all error text messages for the form:

```
Do $cinst.$clientcommand("clearerrors")
```

equivalent to assigning $errortext to empty for all controls on the form which have error text.

Changing the appearance of error text

The following CSS classes control the appearance of the error text and border. These classes are stored in the core.css, which can be modified by overriding them in user.css.

- **div.om-error-text**
  This class styles the error text.
- **div.om-error-border**
  This class styles the outer div of a control which has error text.

Field List

The **Field List** provides a list of all the components on a remote form, including all those controls in container fields, and is often useful if you need to select a specific component which is partly obscured or hidden in the form, or is hidden deep inside the object hierarchy. To open the Field list, select the **Field List** option from the form or component context menu (Right-click to open), or press the **F7** key.

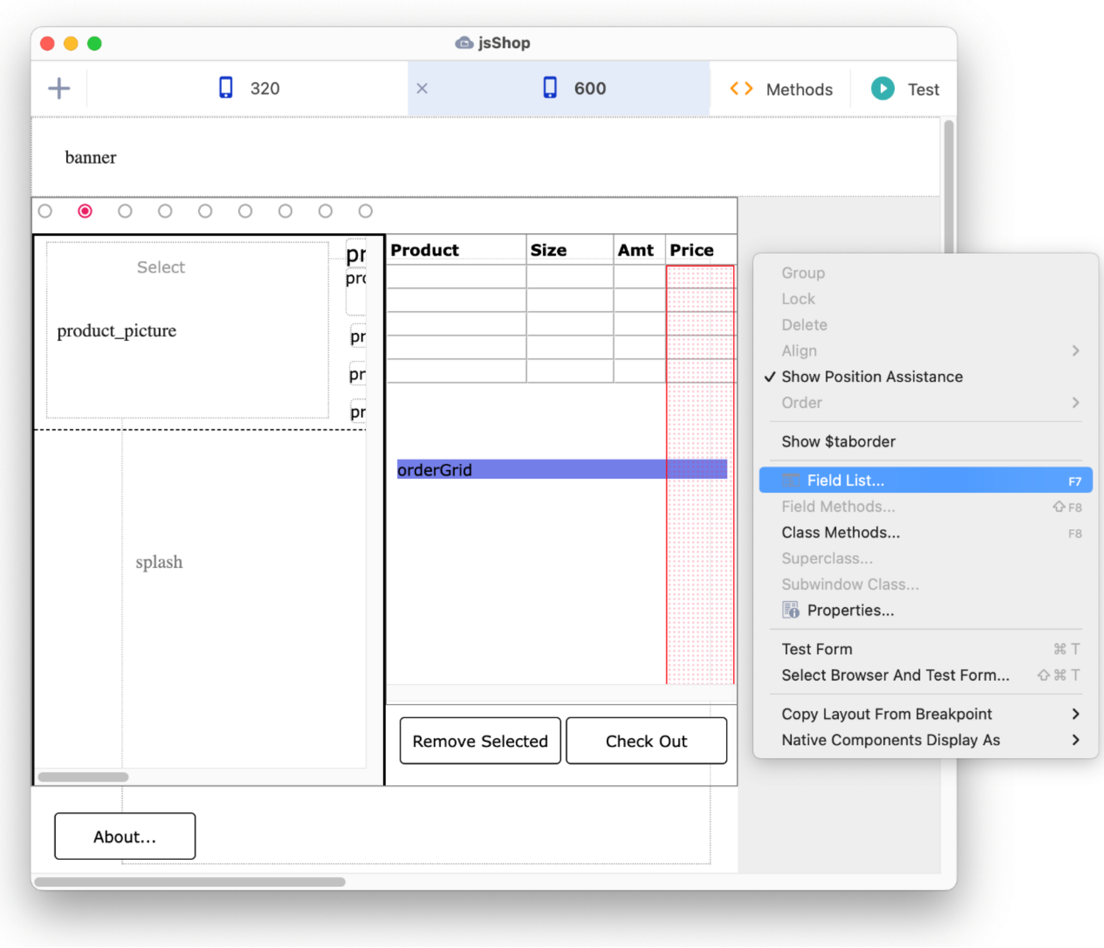


Figure 106:

The currently selected component is checked in the Field list, expanding the tree nodes and scrolling the list if necessary. Conversely, you can open the Field List and check a component name in the list to select it in the form. For example, in the following screenshot, the Order Now button has been checked in the Field List and the corresponding control is selected in the remote form, scolling the design screen if required to locate the control.

When the focus is on the Field List, you can use the arrow keys to navigate up and down the list and use Space bar to select a control, as required. The Shift-Space keypress allows you to select (or deselect) multiple, discontinuous controls in the list.

The Field list can be useful if you need to select the background of a form, for example to set its properties in the **Property Manager,** but the form is completely filled with components and no form background is available to click on, as can be the case for mobile forms. To select the form in this case, you can open the Field List and check the form name at the top of the list (which will deselect

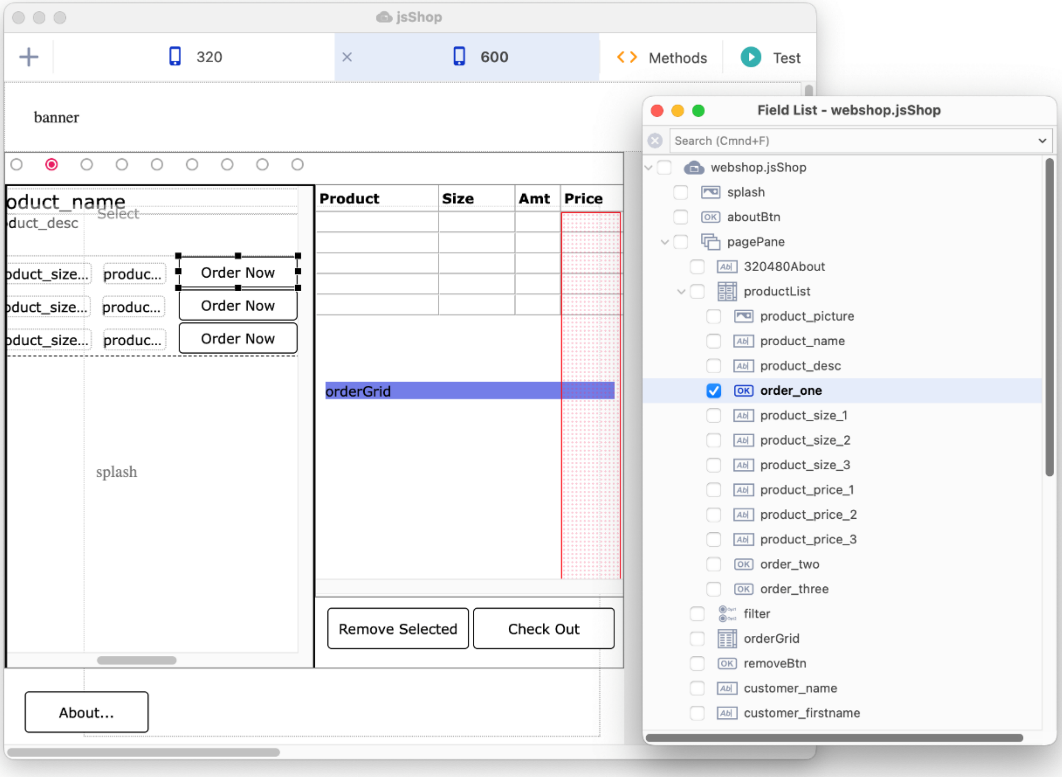


Figure 107:

any components), or if you click on any individual component, then shift-click it to deselect it, the focus will be returned to the form background and its properties will be shown in the Property Manager.

Searching the Field List

The **Search** at the top-left of the Field List is useful if your remote form contains many nested objects, or you want to search for objects with a specific prefix. The search looks for items *containing* the search string. The following shows all objects in the Field list containing the string 'prod'.

Renaming Objects in the Field List

You can rename a component or background object directly in the **Field List,** either using the **Rename** option in the Context menu, or by clicking into the selected line, or by selecting the line and pressing Return to select the existing name. The $name property for the object in the form is updated automatically.

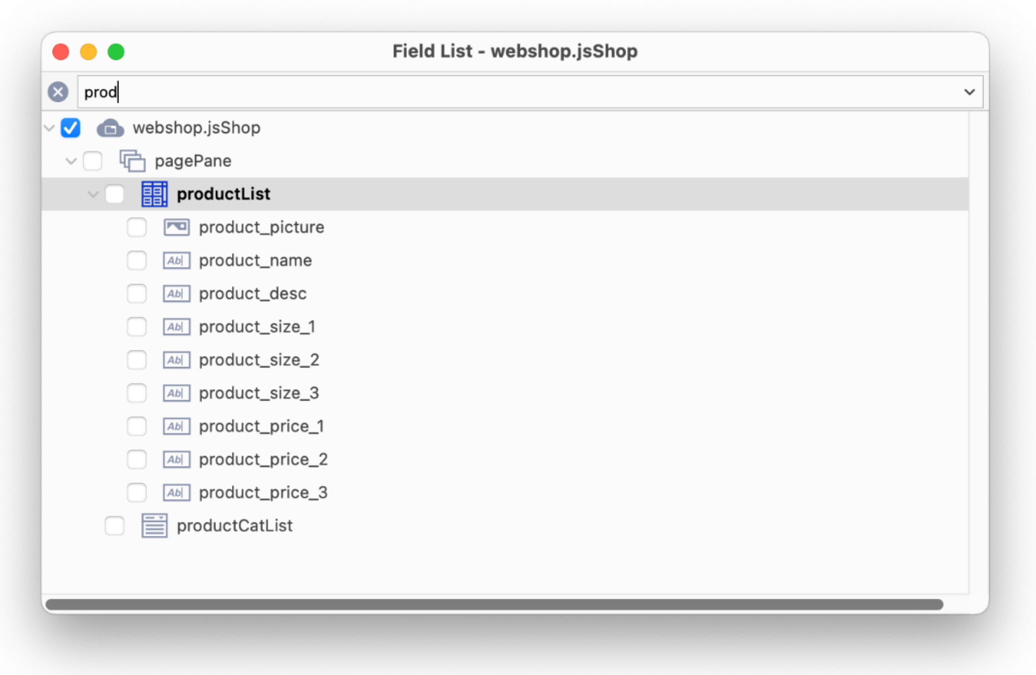
When you rename an object on a remote form (using the Property Manager or Field List), Omnis searches for any properties using the old name, and replaces them with the new name, including properties such as $arialabelledby and $linkedobject.

Component Events

Most of the JavaScript components report events which you can handle in a special method called an "event method" which is inserted behind the component. The event method for a component must be named $event. For example, when the end user clicks a button, an evClick is generated which you can trap in the $event method for the button; this method could contain code to display a message, call another method or determine some other action depending on the code in the event method itself. Most of the components contain a template $event method with a code stub for you to use as a starting point to handle the event.

Enabling Events

To report an event, the event must be enabled for the component. Many of the components have their events enabled by default, but for some you may need to enable specific events in the $events property for the component.

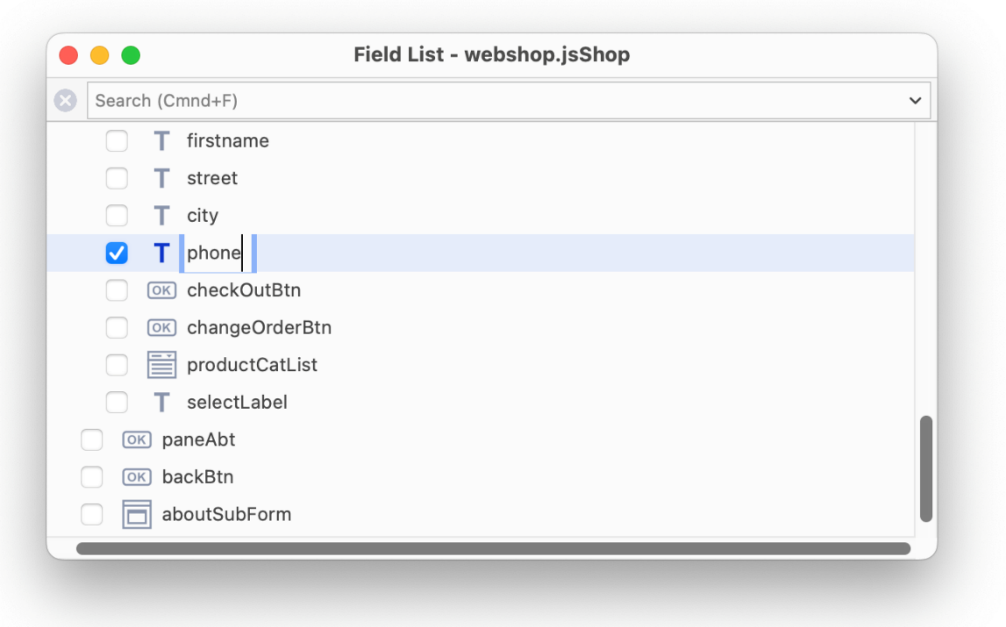


Field List - webshop.jsShop

prod

- ☑ webshop.jsShop
  - □ pagePane
    - □ **productList**
      - □ product_picture
      - □ product_name
      - □ product_desc
      - □ product_size_1
      - □ product_size_2
      - □ product_size_3
      - □ product_price_1
      - □ product_price_2
      - □ product_price_3
    - □ productCatList

Figure 108:



Field List - webshop.jsShop

Search (Cmnd+F)

- □ firstname
- □ street
- □ city
- ☑ phone
- □ checkOutBtn
- □ changeOrderBtn
- □ productCatList
- □ selectLabel
- □ paneAbt
- □ backBtn
- □ aboutSubForm

Figure 109:

To enable an event

- Select the component and open the Property Manager (press F6)
- Click on the $events property in the Property Manager to drop down the list of events for that component (the property will show "No Events" when no events are selected)
- Check (enable) the events you wish to trigger for this component

You can select multiple objects of the *same type* and specify the events for all of the objects at the same time. For example, you can select a number of check boxes and enable the evClick event under the $events property to enable the event for all the selected check boxes.

Editing Event Methods

If you double-click a component in design mode, the Method Editor will open displaying the method for that individual component. For components with events, the $event method will be shown automatically. For example, if you double-click on a button, the Method Editor will open displaying the $event method containing the code *On evClick;* you can add more code after this line to be run when the end user clicks the button. See the example code for each component for example $event methods.

For some components, the $event method may not contain any template code including the *On event* command, but you can add you own. You will need to enter the *On* command and select the appropriate event code (a constant beginning ev) from the Helper window in the Code Editor.

Event Method Validation

Omnis validates the event codes you have entered when adding or editing *On* event commands in the Code Editor. Therefore, Omnis will check to see if the event code is valid for the current object, and if not, it will flag it as an error.

If the event is not specified in the $events property, Omnis will add it to $events automatically when editing a method named $event in a non-inherited object (Omnis displays a temporary status bar message when it does this).

You can turn off this validation using the **validateEventsForOnCommand** entry in the methodEditor group of config.json; set it to false to turn off event method validation.

Standard Field Events

Most JavaScript fields or controls report the evBefore and evAfter events, which are triggered when the focus is about to enter or leave the control, respectively. Note that for edit controls, if the data does not change then an evAfter is not triggered as the focus leaves the control.

You can use the On event command to detect events in your event handling methods, and for most controls the $event method will contain a template event handling method into which you can add your own code. For example, in the $event method for an Edit control you could use the following commands to detect the evBefore or evAfter event.

```
On evBefore
  # do something..

On evAfter
  # do something else..
```

Buttons and Lists

Buttons and all the list type JavaScript controls report the evClick event, as well as evBefore and evAfter; and some list types also report the evDoubleClick event. For example, the Data Grid control reports evDoubleClick which you could detect and initiate a search based on the content of the grid line clicked on.

evAfter event queue

When an event is being executed in the JavaScript client, such as a click on a button, a transparent overlay is applied to the whole remote form, to prevent user interaction anywhere else in the form **and** to maintain the Omnis event ordering. If the user clicks on this overlay, the click will be prevented, although most events happen almost instantaneously so in this case the overlay is not displayed.

For evAfter events that show the overlay, Omnis shows a feedback effect at the point of the click when the overlay prevents the click, to make it clear to the user that their click was not registered. The feedback effect is a No Entry icon, with "bubble" animation, that appears and disappears directly after the user click. In this case, the click will be queued and will fire once the overlay is removed.

Unfortunately, Firefox does not treat the active state of elements in the same way as other browsers. As such, it was not possible to implement these changes for that browser.

Drag Border Event

All JavaScript controls report the evDragBorder which is triggered when the border is dragged – the event is reported to the control containin the border being dragged and any that share the border area being dragged. When it is triggered it could mean that the end user has resized the field (and therefore other fields in the same parent have resized) using the drag border.

Component Icons

Some of the JavaScript Controls, such as the Button control, allow you to add an icon to create a better appearance and UX for your apps. Such controls that support icons have the **$iconid** property which allows you to specify an icon image file to be used for the control.

You can use SVG image files for JavaScript component icons, as well as PNG image files (supported in versions prior to Studio 10.2). You can use SVG image files in an icon set, alongside an existing icon set containing PNG files, and these will appear in the Select Icon dialog when you assign an icon to a JS component.

In addition to single component icons, you can add notification or **'Icon Badges'** to JS component icons to provide additional information, such as a number count: see Icon Badges.

Selecting an icon

You can assign an icon to a JavaScript Component by setting its **$iconid** property in the Property Manager under the Appearance tab: when you click on the dropdown menu for the $iconid property the **Select Icon** dialog opens, allowing you to select an icon image from those available in Omnis.

When you first open the Select Icon dialog you should see the **'material'** icon set from which you can select an icon; the 48x48 icon size may be selected under the **Size** list, but you can select a different icon size, including 16x16 or 32x32, or you can set a custom size. When specifying $iconid in the Property Manager, the id edit field allows you to enter the size of an SVG icon by entering iconid+wxh, e.g. to set an alarm icon with a width of 22 and height of 33, you can enter alarm+22x33.

You can scroll the list of material icons to find an icon you need, or you can enter a search string into the **Search** box to filter the list. For example, the following screenshot shows the 48x48 material icons and the search 'add'.

You can add your own icons to the Select Icon dialog by adding your own icon set; see below. You should use scalable SVG icons for your web and mobile applications, which will be displayed correctly in high definition on phones, tablets and HD monitors. In most cases, you should add your own icon set, tailored to the unique functionality or style of your application, and for all new applications you are advised to use SVG images for icons.

You can use the icons in the 'studio' icon set but these are PNG image files and may not lend themselves to your application, so you are advised to source your own SVG icons.

IMPORTANT: You should not use any icons listed under 'Omnispic', 'Userpic', '#ICONS', or 'IPHONECONTROLS' since these only contain low definition or non-alpha image files and are only present for backwards compatibility for older applications.

Icon sets

SVG icon image files must be stored in an **icon set,** which is a sub-folder of the 'iconsets' folder in the main Omnis tree (do not use the 'icons' folder which contains the legacy icon datafiles, such as Omnispic). Note that an icon set cannot be named 'datafile', 'lib', 'studio', or 'studioide' since those names are already in use and would cause a conflict.

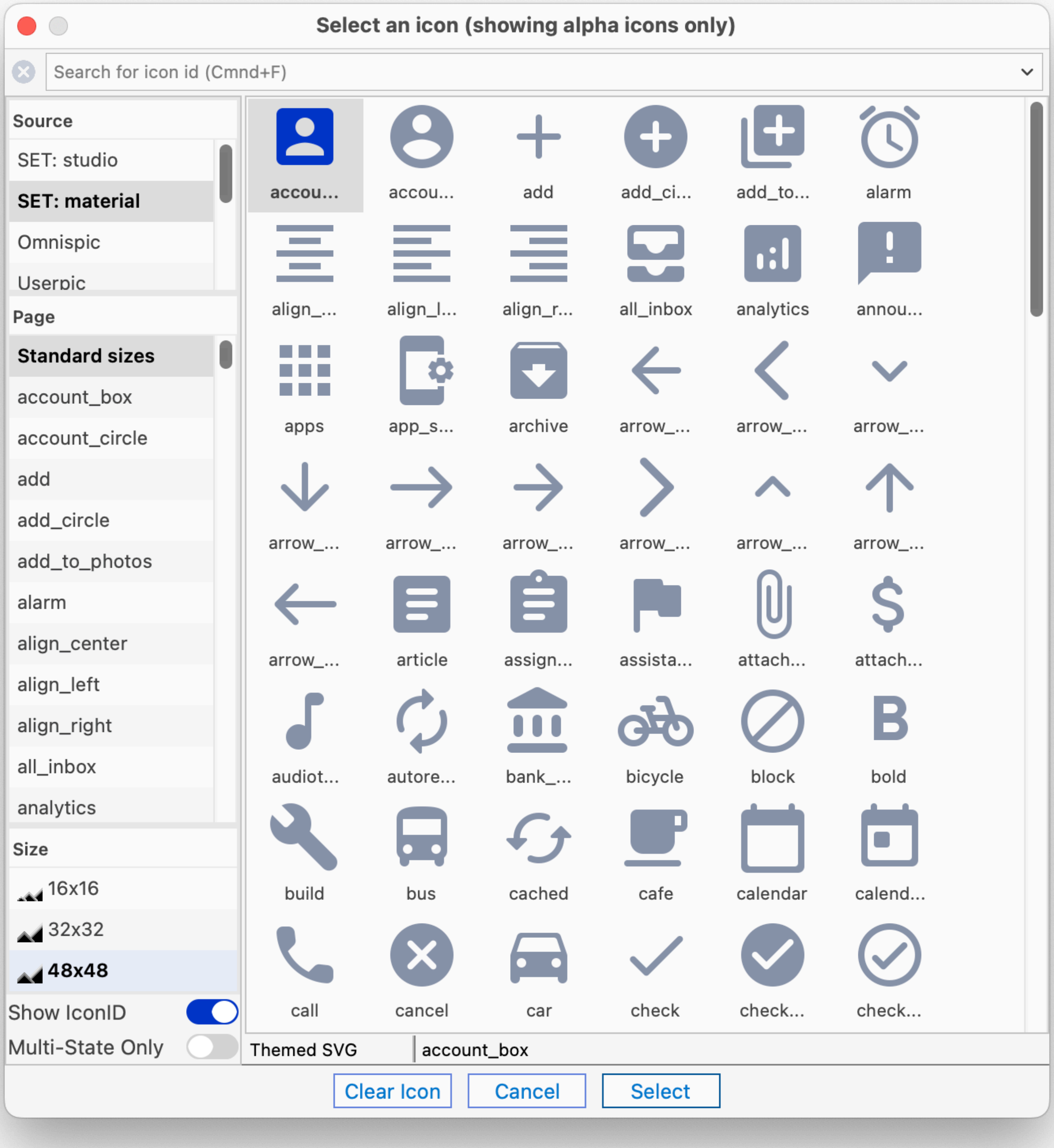

Select an icon (showing alpha icons only)
Search for icon id (Cmnd+F)
Source
SET: studio
SET: material
Omnispic
Userpic
Page
Standard sizes
account_box
account_circle
add
add_circle
add_to_photos
alarm
align_center
align_left
align_right
all_inbox
analytics
Size
16x16
32x32
48x48
Show IconID
Multi-State Only
accou...
accou...
add
add_ci...
add_to...
alarm
align_...
align_l...
align_r...
all_inbox
analytics
annou...
apps
app_s...
archive
arrow_...
arrow_...
arrow_...
arrow_...
arrow_...
arrow_...
arrow_...
arrow_...
arrow_...
arrow_...
article
assign...
assista...
attach...
attach...
audiot...
autore...
bank_...
bicycle
block
bold
build
bus
cached
cafe
calendar
calend...
call
cancel
car
check
check...
check...
Themed SVG
account_box
Clear Icon
Cancel
Select

Figure 110:

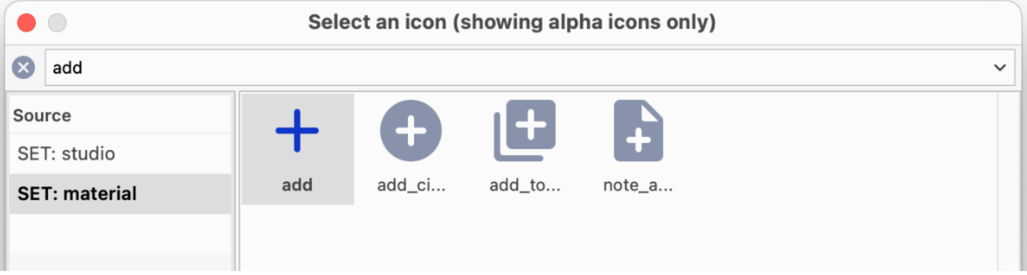


Figure 111:

In order to use any icons in an icon set, you need to add the icon set name to the **$iconsets** library preference, which is a list of icon sets for the library. Once you have added an icon set to $iconsets for a library, the icons will appear in the Select Icon dialog, prefixed SET. (Note this library preference was called $iconset in versions prior to Studio 10.2 and only allowed a single icon set to be used.)

Icon Search Order

Icons can potentially be stored in various locations in Omnis including the 'studio' icon set, as well as any icon sets you have added, plus the various icon data files used in older versions of Omnis: this may become an issue where duplicate icon names or IDs exist across the different locations, so Omnis employs a specific 'icon search order' that determines how icons are located and displayed. If an icon with the same name or ID is included in another folder, after an icon has been found, it is ignored in subsequent folders and an error is written to iconsetlog.txt. You should therefore avoid having the same icon names, IDs, or icon set names in multiple folders to avoid any potential confusion. Omnis looks in the following icon sets or datafiles in this order:

1. The icon set(s) specified in the $iconsets library preference, in the order listed in the property

2. #ICONS for the library, if used (would only be the case for older applications not using SVG or HD PNGs)

3. User icon datafiles (other than Omnispic and Userpic), if used; this is for legacy apps only

4. The 'studio' icon set, under the 'iconsets' folder

5. Omnispic or Userpic (.df1 files located in 'icons' folder), if used

When using a web server for deploying your web or mobile application, any icon sets used in your library must be placed in the 'html/icons' folder in the web server tree, even if they are in one of the other folders in the Omnis Server tree.

During SCAF generation (for the serverless client), the Omnis Server now passes all the files for all icon sets in $iconsets to the serverless client library.

SVG Icons

SVG images are vector based and are inherently scalable, therefore a single SVG file can provide multiple sizes for icons: in practice, an SVG icon will scale to fit the icon area available in a control, such as a button (unless you fix its size, see below). By contrast, PNG images have a fixed size and therefore you have to create a separate image file for each icon size or resolution you wish to support and place all the separate files in an icon set in the Omnis tree. In addition, a single SVG image will have a much smaller file size than mulitple PNG files, giving your app a smaller footprint on the client.

On macOS, SVG icons only render in the thick client when using macOS 10.13 or later.

On Windows, SVG icons only render when using the Windows 10 Creators Update or later. In general, support for SVG in Windows is more limited than on macOS, for example, Windows does not support classes in SVG files – read here about Windows SVG support:

https://docs.microsoft.com/en-us/windows/win32/direct2d/svg-support

Material SVG icon set

Google provides a large set of SVG icons in its *Material design scheme,* which are issued under the Apache License Version 2.0:

(https://fonts.google.com/icons?icon.set=Material+Icons)

You are free to use these in your Omnis applications with the proper attribution in your product licensing.

We have selected over 100 of the Material icons (from the “black rounded” style) and placed them in an icon set folder called ‘material’ under the ‘iconsets’ folder in the Omnis tree. The material icon set will appear in the **Select Icon** dialog by default, and you are free to use these in your applications; the following screenshot shows some of the material icons, with the 16x16 size selected.

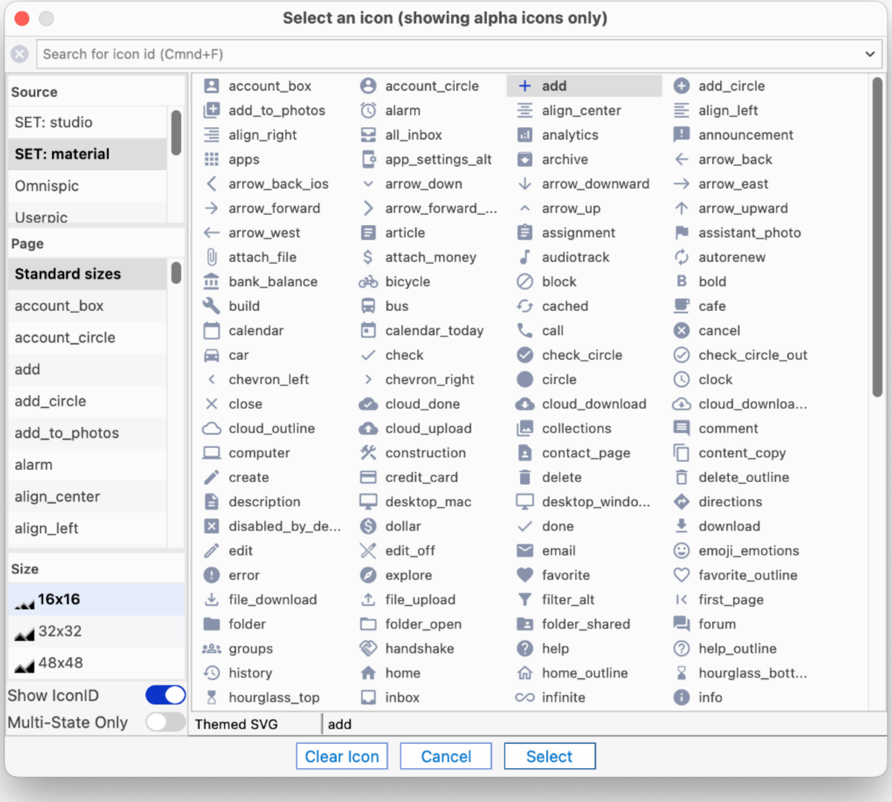


Figure 112:

These material icons have been ‘themed’ using the Omnis SVG Themer tool and therefore support JS Themes. You could download other icons from the Material website and add them to this folder, if required, or create your own new iconset, and use the SVG Themer tool if you want them to support themes (in the Tools>>Add Ons menu). See the Themed Icons section about how to theme icons.

Using SVG Icons

If a JavaScript component can support SVG icons (and most do), then the icon names of any SVG icons will appear in the Select Icon dialog when you assign the icon via the Property Manager and the **Select Icon** dialog (if a component does not support SVG icons, then they are not shown in the Select Icon dialog).

In general, SVG icons are supported by any controls that previously required an icon, including the following classes or features:

- Remote Form class components (JavaScript Client controls), including buttons, menus, toolbars, lists, tabs, check boxes
- Styled text, including styled text on reports sent to the Omnis PDF report destination
- The background icon for the main Omnis window on the Windows platform ($root.$prefs.$backgroundiconid)
- The $componenticon class property

You should note the following for JS controls only:

- Some JS controls use background-image CSS, so when using an SVG image, it will not always scale as expected if the aspect ratio in the SVG is fixed, and the desired dimensions of the background-image do not have the same aspect ratio.
- JS Popup menu and JS Navmenu controls have hot iconid properties – in this case, the hot and equivalent non-hot iconid properties must either both use SVG or both use PNG

Creating SVG Icons

You can create your own SVG icons, or you may be able to acquire a set of icons from a third-party, either paid-for or for free (subject to the appropriate licensing). SVG image files must be saved with the .svg file extension (see naming below) and should be placed in an icon set in the 'iconsets' folder in the Omnis tree, and the icon set name needs to be added to the list of icon sets in the $iconsets preference in your library.

From our testing, we found that Adobe® Illustrator® allows you to export vector images in SVG format, and on the export to SVG options dialog you can select the 'Inline Style' option to ensure classes are not used in the output SVG. There are many other image editors that can output SVG.

SVG icon file names

The base icon ID of an SVG icon is the name of the SVG file, without the file extension, and converted to lower case, up to a maximum of 32 characters. The naming restrictions for SVG icons are as follows:

- The base icon ID *must not represent an integer* (the icon ID had to be an integer for PNGs, but does not have to be for SVG image files)
- The base icon ID *must not contain the characters* + # , ; = ? (plus, hash, comma, semicolon, equals, or question mark); note + is used to add a size restriction, see below

An icon ID or name can now be either an integer or a string, and integer icon IDs work exactly as they did before (the naming of PNG icon images remains the same).

You cannot use the same file name with different case in an icon set folder, plus it's always good practice to make icon IDs or names unique across different icon sets, since the icon with the first instance of a specific icon ID or name is used.

Any errors related to the naming requirements are written to the icon set log file, which is in the folder logs/iconsets, in the data part of the Omnis tree.

Multi-state SVG Icons

If you want to include icons for different states of a control (for example, checked, highlighted, and checked highlighted for a check box control), you can include separate SVG files with a suffix in their name:

- _c for checked
- _h for highlighted
- _ch for checked and highlighted

For example, SVG files for a check box could include the files: checkbox.svg (for the unchecked icon), checkbox_c.svg, checkbox_h.svg and checkbox_ch.svg (for the different states). These 4 files all result in a single icon with id 'checkbox', and Omnis will select the correct SVG file according to the state of the checkbox.

Fixed and Custom Icon Sizes

An SVG icon will always expand to fit the available space within a control, but it is possible to fix or restrict the size of an icon by adding size information to the end of the icon ID name. The size information has the syntax +<w>x<h> where <w> is the integer width and <h> is the integer height. For example, an SVG icon ID could be any of the following:

- testsvg (unrestricted size)
- testsvg+16x16 (restricted to 16x16, for example, for a menu)
- testsvg+32x48 (restricted to 32 wide x 48 high)

When selecting an SVG icon, the size list includes the configured sizes from config.json, and the current size of the icon, in addition to the standard sizes and kDefSize. There is a + button in the heading of the size list that allows you add a new size. There is an option on the dialog to add the new size to config.json.

The 'customSizes' item in the 'svg' section of config.json allows you to add other sizes. The size list in the Select Icon dialog will show any other sizes specified in the config.json file:

```
  "svg": {
  "customSizes": \[
    "256x256",
    "64x64",
    "128x128"
  ]
}
```

When a custom size is selected in the size list for a full page SVG icon, in addition to the + button, there is a - button which you can use to remove the size from the list, and optionally remove it from config.json.

Omnis uses the default width and height specified in an SVG file to determine the aspect ratio of the icon image. To obtain this, Omnis looks for the width and height attributes of the svg element in the SVG file and uses these if present. If width and height are not present, Omnis uses the viewBox attribute of the svg element to determine the aspect ratio. In this case, you can add a size using the + button in the Select Icon dialog, and use the Keep Aspect Ratio option, to fix the aspect ratio.

Icons for Lists

Certain controls, such as the Icon Array, use a list column to contain an icon ID. To make use of SVG icons, this column needs to be defined as Character. Where you use a mixture of SVG icons and existing PNG icons, the icon IDs can be specified as strings or integers as appropriate.

Icon Caching

You can control the cache size for all icon sets (using PNG and SVG icons) in config.json using the maxCachedIconSetBitmaps entry. This is an integer, which defaults to 1000 bitmaps. If Omnis needs to create a new bitmap for an icon from an icon set, and the current number of cached bitmaps is at this limit, Omnis will free up the least recently used bitmap.

Assigning a URL for images

When you set the $iconid of a JavaScript control you can also assign a URL. In server methods, if the value being assigned is a character value that contains a "/" character then Omnis treats it as a URL generated by the iconurl function (meaning that it can contain alternative icon files for the different client resolutions, and also that the server will pick the correct icon for the client resolution).

In client methods, if the value being assigned is not an Icon ID (a literal integer or integer + icon size constant) then Omnis treats the value as a URL generated by the iconurl function on the server, and the client picks the correct icon for its resolution.

You could generate the required URLs with iconurl() (see below) in the $construct() method of your remote form, and store them in an instance variable list which could then be used in client executed code to assign the correct image to each object.

Image handling for tree lists

For the JavaScript Tree control, the iconid column is an iconurl column, and the $iconurlprefix property is redundant although existing libraries that use $iconurlprefix will continue to work. Instead, the iconurl column should be defined to be of type character, and it should be populated using a server-only function, iconurl(iconid), which returns a URL string containing the name of the image file or a semi-colon separated list of file names if an icon exists in more than one resolution. This enables the client to pick the correct icon for its resolution.

Exporting Icons from an Icon Datafile

You may want to use some existing icons located in an Icon Datafile and either add to or replace some of them with higher resolution versions. To enable you to export existing icons as separate files, there is a tool in the Tools>>Add Ons menu, called the 'JS Icon Export' tool, which is available in the 'Web Client Tools' dialog (scroll to the bottom of the list of Web Client tools). The 'JS Icon Export' tool will export all the icons in a selected Icon Datafile and place them in a folder in the 'iconsets' folder, applying the correct image file names. The $iconid property of a control will now reference the external image file in the icon set and not the icon datafile image, since Omnis looks in the iconset folder for the library before any icon datafiles. The Icon Export tool will only export icon images that support Alpha, i.e. the icon page containing the existing icon(s) must be set to Alpha.

Icons Folder Name

Apache often redirects a URL with "/icons/" to the /usr/share/apache2/icons folder, and you would then need to place all the icons for your app in that folder. Therefore, if you deploy your web or mobile app to an Apache server, you may want to rename the 'icons' folder in Omnis by editing (adding) an entry in the Omnis configuration file (config.json). The "iconsFolder":"omnis_icons" configuration item in the server group of config.json defaults to "icons" if omitted or is empty, so you can change the name by adding your own value. You are recommended to use the same value for development and runtime, since the folder name is stored in the HTML for each remote form class.

PNG Icons

From Studio 10.2 onwards, you are advised to use SVG images for component icons, although you can still use PNG images. In this case, you should create PNG image files that are 16x16, 32x32, or 48x48 pixels either at a standard pixel density suitable for displaying on standard monitors, or image files that are 1.5 and 2 times the size, suitable for displaying on phones and other HD devices. When your app is displayed on different devices and screen resolutions, Omnis will display the correct icon size and resolution.

PNG Image File names

Each PNG image file within an icon set must conform to the following naming convention:

```
<text>_<id>_<size>_<state>_<resolution>.png
```

- <text> is a string, i.e. the name of image, which must not contain underscore. This string is used in the icon picker dialog when you set an object's $iconid in the Property Manager so it should describe the icon. Icon files that are the same image, but different resolutions should have exactly the same <text> name.
- <id> is the positive integer id to be used as the icon id. It can be in the range 1 to 10000000. Icon files that are the same image, but different resolutions should have exactly the same <id>.
- <size> is the CSS pixel size of the image, i.e. the resolution independent size of the image, meaning that for all resolutions of the same image this has the same value.

The value of <size> has the form <width>x<height>, where the values 16x16, 32x32 and 48x48 are special values since they correspond to the standard icon sizes supported by Omnis.

- <resolution> is the factor by which the pixel density is greater than a standard monitor and is one of the following:
  "_2x" for HD devices such as the Retina display
  "_15x" for some devices e.g. certain Android phones that have a 1.5x pixel density.
  an empty string is the default and is for standard resolution devices, equivalent to _1x

Any files (or folder names) that do not conform to the naming conventions are ignored.

Note that the image file names are case insensitive and they must be unique across all platforms and file systems (that is the case of file names is ignored).

If you are unsure about the file naming for PNG icons, you can examine the icons in the 'iconsets/studioide' folder.

PNG Check Boxes Icons

You can use PNG images for check box and radio button icons, using the following naming:

- <state> is the checked, highlighted, or normal state of the icon for multi-state icons and can be one of the following:
  an empty string for the normal state of the icon
  “c” is the checked state of the icon
  “h” is the highlighted state of the icon
  “x” is the checked highlighted state of the icon

PNG Image Scaling

You do not have to create an icon PNG image for all resolutions, although it would be advisable to do this for the best appearance. Omnis will use an icon image closest to the resolution being referenced, scaling as appropriate, and as with all image scaling it is better to force Omnis to scale an image down rather than scale it up. In this case, you may like to provide the highest possible resolution image for your icons and allow Omnis to scale the images down to display an icon for lower resolutions, but the scaling may produce unexpected results.

When the JavaScript Client connects, it sends its resolution to the Omnis App Server. This allows the server to use the appropriate icon when setting iconid properties in server methods.

Non-standard PNG image sizes

You can create PNG images with a size other than the standard sizes (16x16, 32x32, 48x48) by creating the image at a non-standard size and including the image size in the file name when the file is saved. For example, you can create an image 100x200 pixels and name it something like “mygraphic_1688_100x200.png”, and you can create a high resolution version at 200x400 pixels and name it “mygraphic_1688_100x200_2x.png”. (This is the equivalent of an ‘Icon Page’ in older versions of Omnins.)

Icon Data files and #ICONS

NOTE TO EXISTING USERS: The method of storing icons in #ICONS or an Icon data file (such as Omnispic) and assigning the numeric Icon ID ($iconid) to controls will continue to work, but this is only useful for icon images that are 16x16 pixel (or 32x32 for high def). In this case, if you run your application on an HD display and your library uses an icon data file or #ICONS, Omnis will try to use a 32x32 icon (if it exists and the icon page is marked as containing 32x32 icons), in place of the corresponding 16x16 icon. If a 32x32 image does not exist in your icon data file or #ICONS, the existing 16x16 image will be used which may have a very poor visual appearance on newer screens and devices. In order to support high definition 16x16 icons you will need to create a new version of each image at 32x32 pixels and import each one into the icon data file or #ICONS into the 32x32 section on the same icon page using the same icon IDs.

If you have used 32x32 or 48x48 pixel icons in your libraries (in #ICONS or an Icon data file), and you wish to display them on HD displays, then you will need to adopt the use of icon sets, which support icon images up to 96x96 pixels, that is, 2x the largest 48x48 icon size. Note that icons in an icon set will take precedence over icons in #ICONS, Omnispic or Userpic in the icon search order.

Icon Badges

You can add notification badges or **‘Icon Badges’** to JavaScript component icons to provide additional information, such as a number count, or to alert the end user, in order to enhance the UI in your applications. (Note you can also apply icon badges to Window class component icons.)

Icon badges are additional icons or notifications that can be added to any JavaScript component icon, that is, a badge can be added to any control that supports icons, such as Push buttons, Toolbar buttons, Menu items, or Tab bar tabs. The following screenshot shows some examples, including button icons, toolbar icons, and tabbar icons.

When assigning to $iconid for a JavaScript component, you can use the **iconidwithbadge()** function to assign an icon badge or number count notification and its properties. Therefore, when an icon ID uses an SVG icon name, iconidwithbadge() allows you to append additional values to the SVG name to define a badge to be added to the main icon. The syntax is:

```
iconidwithbadge(svgIcn, count_or_secondary_icon [ , badge_options, backcolor, icontextcolor ] )
```

The parameters are:

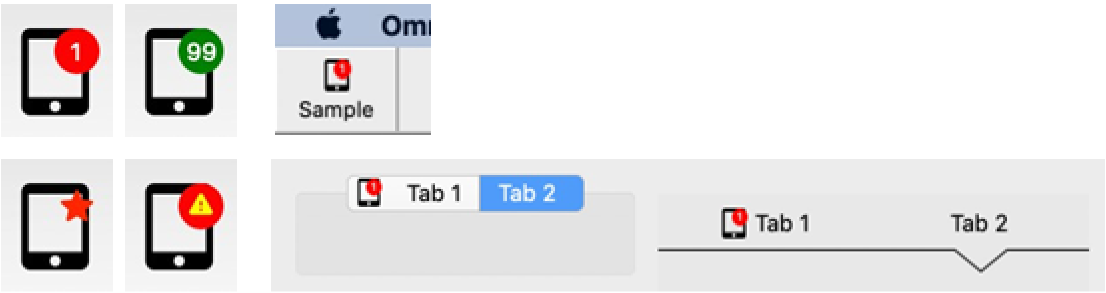


Figure 113:

- **svgIcn**
  the ID of the primary icon for the object / toolbar object
- **count_or_secondary_icon**
  the count to be displayed on the badge, or the ID of a smaller secondary icon
- **badge_option**
  kIconBadgeAlignTop, kIconBadgeAlignBottom, or the default is the position set by the OS, also kIconBadgeBackgroundHide, see below.
- **backcolor**
  the color of the badge, the default is kJSThemeColorSecondary
- **icontextcolor**
  the color of the count, or secondary icon, the default is kJSThemeColorSecondaryText

For example, the following lines of code set up icon badges for buttons:

```
Do $cinst.$objs.button.$iconid.$assign(iconidwithbadge( ‘tablet_mac’, 9 ))
Do $cinst.$objs.button.$iconid.$assign(iconidwithbadge( ‘tablet_mac+32x32’, 9 ))
Do $cinst.$objs.button.$iconid.$assign(iconidwithbadge( ‘tablet_mac’, 99, 0, kDarkGreen, kWhite ))
```

Some Omnis objects used fixed icon sizes, such as menu items or tabbar tabs, therefore when applying a badge to these objects you cannot supply an icon size for the primary icon as the size will be fixed by the object, for example:

```
Do $imenus.NewMenu.$objs.Item.$iconid.$assign(iconidwithbadge( ‘tablet\_mac’, 9 ))
```

When using iconidwithbadge() in a client-executed method, the SVG parameters must be URLs, which can be generated with iconurl() in server-executed code.

The default icon badge background colour is kJSThemeColorSecondary, while the count or secondary icon is kJSThemeColorSecondaryText (for window class controls the colors are the standard OS colors).

Badge Options

The constants **kIconBadgeAlignTop** and **kIconBadgeAlignBottom** can be used in the badge_option parameter in iconidwithbadge() to specify the position of the badge. Omitting this or passing 0, Omnis will use the default position for the OS – by default, macOS will draw a badge at the top right of an icon, and Windows at the bottom right.

The constant **kIconBadgeBackgroundHide** allows you to hide the default colored circle badge when used with a secondary icon. If the badge has a count and not an icon, the badge background is always drawn and this option ignored. For example:

```
 $iconid.$assign(iconidwithbadge( 'tablet\_mac', 'star', kBadgeIconHideBackground, kDefault, kRed ))
```

Tab panes and Tab strips

To set an icon badge on a tab pane or tab strip, you can use a new method **$settabinfo()** – this allows you to alter a tab name or icon at runtime without first changing the current tab. The syntax is:

```
$settabinfo(** *tabnumber*, *caption*, *icon* )
```

The parameters are:

- **tabnumber**
  a valid tab from 1 to $tabcount
- **caption**
  the new tab caption or empty to leave caption untouched
- **icon**
  the icon for the tab; you can use iconidwithbadge()

The new iconidwithbadge() function can be used to specify the icon badge. For example:

```
Do $cinst.$objs.tabpaneorstrip.$settabinfo( 1, '', iconidwithbadge( 'tablet\_mac', 1 ) )
```

Component Fonts

The font for all JS controls is set using the **$font** property. The **Roboto Flex** font is the default font for all JS components (in new libraries), including Entry fields and labels. Roboto is a Google font and included in the folder html/fonts; its use is subject to the Apache License Version 2.0: https://www.apache.org/licenses/LICENSE-2.0

The "system-ui" font is also available for most controls and uses the Operating System's default font, so changes between platforms. This may be useful if you are designing a mobile app to run in the wrappers, giving your app a more native look.

The **$fontstyle** and **$fontsize** properties sets the font style or weight, and font size, respectively. This includes semi-bold (kSemiBold) if the font supports it (Roboto does). Using both kBold and kSemiBold causes an extra bold font style to be used.

Drag and Drop Data

Drag and drop for the JavaScript Client provides the ability for end users to drag data from one JavaScript control in a remote form, and drop that data onto another JavaScript control. In addition, end users can drag files from their desktop and drop them onto a JavaScript control within a remote form displayed in their web browser.

IMPORTANT NOTE: Support for drag and drop in JavaScript remote forms is limited to desktop browsers only, including Chrome, Edge, Firefox, IE 11, and Safari – drag and drop is *not supported* in mobile browsers. Also note drag and drop only applies from one control to another control, or a file onto a control – you cannot drag data to or from a remote form.

To drag and drop some data, the end user can click and hold down the pointer over a JavaScript control on a remote form, then drag the highlighted control onto another control and release the pointer when the target control is highlighted. To enable drag and drop, you have to set various properties in the source and target JavaScript controls, and handle various events in each control as the drag and drop events occur.

Existing users should note that the event constants and their parameters work in a very similar manner to those for the drag and drop mechanism in the thick client, with the addition of the pDropId parameter which identifies the area of a control over which the drop is to occur (see under Events).

Example Library

There is an example library demonstrating how you can drag and drop images between JavaScript controls, and the library allows image files to be dropped onto a control from the desktop. The example library is called Drag and Drop and is available in the **Samples** section in the **Hub** in the Studio Browser, in the JavaScript Component Gallery.

Dragging Data

Dragging data is limited to certain data-bound JavaScript controls and is not possible for all types of JavaScript controls. JavaScript client controls that support dragging data will have the $dragmode property. This can be set to either kNoDragging or kDragData.

Note that the $dragiconid property used in the thick client is not supported for drag and drop in the JavaScript client, for a number of technical limitations in various browsers. The dragged image is typically an image of the dragged element created by the browser, using the content of the element when the drag starts – the client performs various temporary adjustments to the element to make the dragged image correspond to the dragged data as appropriate.

Dropping Data

A drop can occur on any JavaScript control, but *remotes forms do not accept drops*. You can specify that a control can accept dropped data by setting its $dropmode property. When a control can accept some data, the JavaScript client highlights the destination control. For JavaScript client controls, $dropmode can be one of the following constants:

- **kAcceptControl**
  Data from a JavaScript client control can be dropped onto this control.
- **kAcceptFiles**
  Files dragged from the system (desktop) can be dropped onto this control; this would allow you to upload the file, for example, which is described in Dragging and Dropping Files

In addition, the list, tree list and data grid controls have the $hiliteline property, indicating that data can be dropped on a specific list line or tree node rather than the entire control. This also means that rather than highlighting the entire control, the client highlights the current destination line or node when a drop can occur.

Scrolling

When the end user is dragging data, they can scroll a destination control vertically by placing and holding the pointer near the bottom or top of the control. This is useful with long lists, grids or tree controls, when the $hiliteline property is enabled.

Events

In order to process a drag and drop procedure, you have to handle some events in the $event method in the source and target controls. The drag and drop events must be enabled as required in the $events property for a control.

evDrag

The client sends evDrag when the user attempts to start a drag. evDrag must be executed in a client-executed $event method, and it has the following event parameters:

Parameter	Description
pDragType	Always set to the value kDragData
pDragValue	Described in the Drag Values section below

If you use *Quit event handler* (discard event) during evDrag, you prevent the drag from starting.

Since it is not always convenient to mark $event for a control as client-executed, the client provides an alternative mechanism. You can implement a client-executed method named $drag for the object, with two parameters (type Var): pDragType and pDragValue. $drag returns true if the drag is allowed, false if not.

The client first attempts to call $drag. If $drag exists and returns true or false, then the drag starts or is not allowed to start respectively. If $drag does not exist, or does not return a value, Omnis sends evDrag if it is selected to execute in $events, and if $event is client-executed.

The drag will only fail to start if $drag executed and returned false, or if evDrag was sent and discarded by Quit event handler.

Data grids, lists and tree controls may select a line or node when the drag starts. This will result in a click event being sent just before $drag is called or evDrag is sent. If the click is sent to the server, it will execute in parallel with evDrag or $drag.

evDragFinished

The client sends evDragFinished when the user has finished a drag (released the pointer). It has no event-specific parameters. evDragFinished can be server or client executed.

evCanDrop

The client sends evCanDrop when the pointer is over a control that can accept a drop of the current drag type (kDragData or kDragFiles). evCanDrop must be executed in a client-executed $event method, and it has the following event parameters:

Parameter	Description
pDragType	kDragData if data is being dragged from a control, or kDragFiles if a file or files are being dragged from the system
pDragValue	Described in the Drag Values section below. Note that if pDragType is kDragFiles, this is empty during evCanDrop, since information about the files being dragged is not provided by the browser
pDragField	If pDragType is kDragData, this contains the name of the field from which data is being dragged. If pDragType is kDragFiles, this is empty.
pDropId	The identifier of the area of the control over which the drop is to occur. Either a line number or ident (when $hiliteline is true), or zero if the control is not list-based (or $hiliteline is false).

If you use *Quit event handler* (discard event) during evCanDrop, you prevent a drop on to the current control and pDropId combination.

Since it is not always convenient to mark $event as client-executed, the client provides an alternative mechanism. You can implement a client-executed method named $candrop for the object, with four parameters (type Var): pDragType, pDragValue, pDragField and pDropId. $candrop returns true if the drop is allowed, false if not.

```
If pDragField=$cobj.$name
  Quit method kFalse
End If
```

The client first attempts to call $candrop. If $candrop exists and returns true or false, then the drop is allowed or not allowed respectively. If $candrop does not exist, or does not return a value, Omnis sends evCanDrop if it is selected to execute in $events, and if $event is client-executed.

The drop will only be denied if $candrop executed and returned false, or if evCanDrop was sent and discarded by Quit event handler.

evWillDrop

The client sends evWillDrop when a drop occurs over a control and drop id combination for which which a drop is allowed according to the can drop processing. The client sends evWillDrop to the control being dragged - therefore, evWillDrop is not sent when dragging files from the system. evWillDrop can be server or client executed. It has the following event parameters:

Parameter	Desciption
pDragType	kDragData
pDragValue	Described in the Drag Values section below.

Parameter	Desciption
pDropField	The name of the control where the data is being dropped.
pDropId	The identifier of the area of the control over which the drop is occurring. Either a line number or ident (when $hiliteline is true), or zero if the control is not list-based (or $hiliteline is false).

Quit event handler with discard event has no effect on evWillDrop.

evDrop

The client sends evDrop when a drop occurs over a control and drop id combination for which a drop is allowed according to the can drop processing. evDrop can be server or client executed. It has the following event parameters:

Parameter	Description
pDragType	kDragData if data is being dragged from a control, or kDragFiles if a file or files are being dragged from the system.
pDragValue	Described in the Drag Values section below.
pDragField	If pDragType is kDragData, this contains the name of the field from which data is being dragged. If pDragType is kDragFiles, this is empty.
pDropId	The identifier of the area of the control over which the drop is occurring. Either a line number or ident (when $hiliteline is true), or zero if the control is not list-based (or $hiliteline is false).

Quit event handler with discard event has no effect on evDrop.

The following $event method is behind an image control and processes the dropped data (this is available in the example library).

```
On evDrop
  If pDragType=kDragData
    If pDragField='LeftImage'
    Calculate iLeftImage as iRightImage
    # pDragValue is base64, convert to binary for consistancy
    Calculate lBase64 as mid(pDragValue,pos(',',pDragValue)+1)
    Calculate iRightImage as binfrombase64(lBase64)
  End If
Else
  Calculate lLine as 1
  # pDragValue can contain many lines use first file only
  Calculate iRightImageIdent as pDragValue.[lLine].4
  Do $cinst.$clientcommand(
    'readfile',row(iRightImageIdent,'iReadFileBin',kTrue))
  # readfile is a client command - see below
End If
```

Drag Values

This section describes both the controls for which data can be dragged, and the drag values generated for each drag.

Combo box

pDragValue is the selected text dragged from the current selection in the entry field component of the combo box. To drag text, you must click and hold the pointer somewhere in the selection before dragging.

Data grid

pDragValue is a list. For a single select data grid, the list has one line, containing the list line being dragged. For a multiple select data grid, the list contains the selected lines being dragged.

Entry

pDragValue is the selected text dragged from the current selection in the entry field. To drag text, you must click and hold the pointer somewhere in the selection before dragging.

List

pDragValue is a list containing the list line being dragged.

Picture

pDragValue is a character string containing the URL of the picture being dragged. If the picture is populated using a variable and $mediatype, the URL is a data URL.

Rich text

pDragValue is the selected text dragged from the current selection in the entry field component of the rich text control; note that this is the plain text without any formatting. To drag text, you must click and hold the pointer somewhere in the selection before dragging.

Tree

The tree only supports dragging when it is in dynamic mode (i.e. when $datamode has the value kKSTreeDynamicLoad). pDragValue is a row containing information about the node being dragged. The row has 3 columns:

Column	Description
ident	The ident of the node
tag	The tag of the node (a character string)
text	The node text

If the $hiliteline property is kTrue for a tree control, and the dropmode indicates that the tree is a potential drop target, the client will expand a node when the pointer enters it while dragging.

Tab control

The tab control contains some special logic that allows you to switch tabs while dragging, if this is the functionality you require. For can drop, it sets pDropId to the tab number of the tab under the pointer (or it sets pDropId to the current tab number if the pointer is over an area of the control which is not a tab). To switch tabs, implement a client-executed $candrop method for the tab control which executes:

```
Calculate $cobj.$currenttab as pDropId
Quit method kFalse
```

Dragging and Dropping Files

In addition to dragging and dropping data from one control to another, end users can drag files from their desktop and drop them onto a JavaScript control in a remote form in their browser. There are two client commands that allow you to process dropped files, using the $clientcommand method.

closefile

The client records file idents (and their JavaScript File objects) in a table. Use closefile to remove the table entry and release resources. You should really do this for every ident passed in the drag value to evDrop, unless you use readfile which removes the table entry after reading the file.

The row passed to the "closefile" $clientcommand has a single column, which is the ident of the file to remove from the table. If you pass a row where the ident is zero, the client removes all entries from the table.

readfile

The readfile client command allows you to read the contents of a file identified by its ident. After attempting to read the file, the client removes the ident from the table, so a call to closefile is not required.

The row passed to readfile has the following structure:

```
row(ident,instance variable name,base64)
```

The columns are as follows:

Column	Description
ident	The ident of the file
instance variable name	The name of an instance variable in the form used to call $clientcommand, that will receive the contents of the file. Note that this is a character string containing the instance variable name, not the instance variable itself
base64	A Boolean. If true, the file is read as base64; otherwise the file is read as text

The JavaScript FileReader which the client uses to read the file operates asynchronously, so a call to readfile starts the file reading process. When the file read is complete, the client calls the client-executed method $filereadcomplete in the form used to call $clientcommand. $filereadcomplete has two parameters:

Parameter	Description
ident	The ident of the file.
error text	Empty if the file was read successfully, meaning that the named instance variable has been populated with the file contents (either as text or base64-encoded text). If not empty, some text describing why the file read failed.

Files dragged from system

For file dragging, pDragValue is only populated for evDrop. It is a list of the files dragged from the system, with columns defined as follows:

Column	Description
name	The file name. Note this is just a name, not the path to the file.
type	The MIME type of the file if this was determined by the browser before passing it to the drop event.
size	The size of the file in bytes.
ident	An integer, unique in the context of the client, that identifies this dropped file. You can use this with the client commands described in the later section about processing files.

Drag and Drop for Thick Client

evCanDrop, evWillDrop and evDrop for the thick client have the pDropId parameter. This is significant when $hiliteline for the control is true, and contains the id of the location in the control where the drop would occur or is occurring, e.g. the list line for a list.

Copying data

The end user can copy data in any control and the content can be returned via a client executed method called "$clipboardcopy". For example, the end user can copy selected rows in a data grid and the content can be returned as tab-separated values. In this case, you can add a $clipboardcopy client method to the grid control to handle the clipboard content. The method can return character data or a list. If it is a list, column 1 must be the MIME type and column 2 must be the content. For example:

```
# $clipboardcopy client method
Do lList.$define(lMime,lContent)
Do lList.$add("text/plain","Copy this as plain text")
Do lList.$add("text/html","Copy this as <b><u>HTML</u></b> instead")
Quit method lList
```

Side Panels

A **Side Panel** is a vertical panel that can be displayed down the left or right side of a remote form (like a sidebar), containing clickable options, such as a menu of options or other content. Side panels are a common UI element in dashboard style designs and allow you to create a more interactive UI for your web & mobile apps. *Note that there is not a separate side panel component, instead many existing JavaScript controls can be marked as a side panel by setting the **$sidepanel** property of the control to kTrue.*

A Side Panel will pop out on the left or right side of a form automatically, *when the end user hovers their pointer over the left or right edge of the form.* Alternatively, a side panel can be opened and closed manually using a button. When a side panel is opened it is animated, so when activated, it will slide in or out.

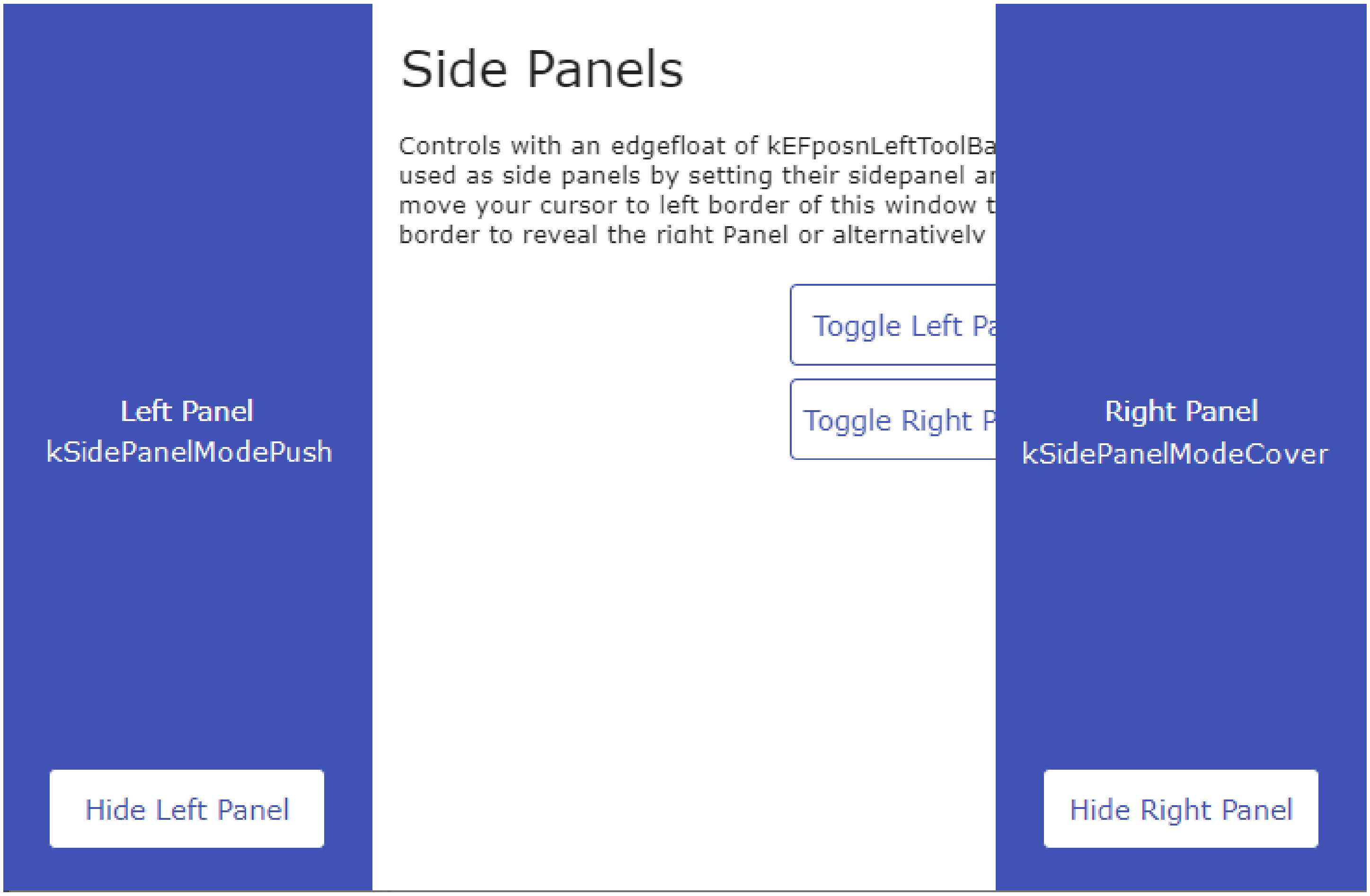
In practice, it would normally make sense to use a container object, such as a **Paged pane, Subform,** or **Scroll Box** as a side panel since you can then add other controls to the container which the end user can interact with. Alternatively, a **Tree list** could be switched to a side panel which would function as a Navigation bar for your web app.

There is an example app called **JS Side Panels** under the **Samples** option in the **Hub** in the Studio Browser, that demonstrates the basic behavior of side panels.

Panel Mode Property

The **$sidepanelmode** property determines the panel mode, that is, how or when the panel is popped out; the mode is set using a kSidePanelMode... constant, as follows:

- **kSidePanelModeNone**
  the default mode meaning the side panel *will not* pop out automatically when the end user hovers over the edge of the form, but the $showpanel method can be used to show the side panel (e.g. executed behind a button)



Side Panels

Controls with an edgefloat of kEFposnLeftToolBa
used as side panels by setting their sidepanel ar
move your cursor to left border of this window t
border to reveal the right Panel or alternatively

Figure 114:

- **kSidePanelModePush**
  the side panel pops out automatically when the end user hovers over the edge of the form and ***"pushes"*** or moves the other controls and content on the remote form either to the right or left

- **kSidePanelModeCover**
  the side panel pops out automatically when the end user hovers over the edge of the form and ***"covers"*** the other form content, i.e. the panel is placed over the top of the other controls and content on the remote form

Panel Mode Method

You can use the $showpanel() method to show or hide a side panel, when $sidepanelmode = kSidePanelModeNone; *the method must be executed on the client.*

- **$showpanel**(*iAction*, [*iMode*=kSidePanelModeAuto])
  Performs an action (*iAction*) on a side panel object, one of the following:
  kSidePanelActionHide hides the side panel.
  kSidePanelActionShow shows the side panel.
  kSidePanelActionToggle either hides or shows the side panel depending on its current state.
  The panel mode (*iMode*) is optional and only applies when iAction is kSidePanelActionShow; if omitted, the default is kSidePanelModeAuto which uses the setting in the $sidepanelmode property, either kSidePanelModePush or kSidePanelModeCover

For example, you could set the $sidepanelmode property to kSidePanelModeNone (i.e. the panel will not pop out automatically), and use the $showpanel() method behind a button to pop it out, as follows:

```
On evClick      ## set to execute on client
  Do $cinst.$objs.panel.$showpanel(kSidePanelActionToggle,kSidePanelModePush)
```

Events

The following events are reported by a component when it is enabled as a side panel.

Event	Description
evWillShow	Sent at the start of the animation when the side panel is about to open
evShown	Sent at the end of the animation when the side panel has finished opening
evWillHide	Sent at the start of the animation when the side panel is about to close
evHidden	Sent at the end of the animation when the side panel has finished closing

evWillShow and evWillHide can only be executed on the client. This is so the events can be discarded, if required, which will prevent the panel from being shown or hidden.

Tab Order

The **$order** component property determines the Tab Order for the controls within a remote form, that is, the order in which remote form controls receive the focus as you press the Tab key. The value of $order for each control is assigned automatically as you add controls to a form in design mode, starting at 1 and increasing by 1 for each control (note the $order values do not change if you move or rearrange the controls on the form). You can change the $order value of a control to change its tab order: when you change the value of one control, the $order value of other controls on the form will be adjusted in sequence automatically.

When you tab into a container, such as a page pane, the tab order takes you through all of the fields in the container, before tabbing out of the container.

The **$startfield** remote form property specifies which field in a remote form will get the focus when the form is opened, overriding the control with its $order property set to 1; $startfield takes the field number as specified in the $order property of the control. Note this property may have an impact on accessibility, insofar as the field specified in $startfield may not be the first field on the form, thereby going against most accessibility practice.

Remove from Tab Order

The **$removefromtaborder** property allows you to remove a control from the tab order. If $removefromtaborder is true, the control is not included in the tab order for the remote form, except for Complex grids which cannot be removed from the tab order. If a control does not have this property, it is always excluded from the tab order, i.e. it cannot be tabbed to.

Design tab order

All JS components have a **$taborder** read-only property in design mode which shows the resolved tab order within the form, taking into account container fields, such as paged panes or complex grids. The context menu on a remote form has a “Show $taborder” option, so that you can see the value of $taborder for all controls on the form. You can still alter the tab order of the controls in a form by modifying $order for each control.

Next Tab Object

All JS components have the **$nexttabobject** property which allows you to override the default tab order, set by the $order property for all the controls in a remote form. The $nexttabobject property allows you to specify the name ($name) of the control you want the focus to jump to after the current object, overriding the tab order set by $order. You can specify the row when setting the $nexttabobject property to a complex grid child.

You should not overuse this property, as it does incur some overhead by setting up additional event listeners.

Accessibility

Omnis Studio supports the Web Content Accessibility Guidelines (WCAG 2.0) which will help to make your applications more accessible, primarily for people with disabilities. These guidelines have been adopted by many government agencies and guarantee an acceptable level of access to information and services via websites and applications for people with disabilities. You can read the following pages to gain a basic understanding of the WCAG requirements:

https://www.w3.org/WAI/standards-guidelines/

The WCAG implementation in Omnis Studio calls on the ARIA specification, which according to W3.org is “**Accessible Rich Internet Applications** (ARIA) defines a way to make Web content and Web applications more accessible to people with disabilities. It especially helps with dynamic content and advanced user interface controls developed with [various web technologies],” which includes technologies such as the JavaScript Client in Omnis Studio.

In practice, most JavaScript components have ARIA compliant properties which you can use in your web and mobile apps to support end users with disabilities. These properties will be read automatically when the screen reader capabilities are enabled in the end user’s browser or mobile device.

ARIA Properties

Most JavaScript controls have a set of basic ARIA and other accessibility properties which are interpreted by the screen reader in the browser. The ARIA properties in Omnis map closely to their equivalent ARIA attributes in HTML.

Several of the JavaScript controls have the following ARIA properties, while some other controls have additional properties (listed below). These properties are designed to work in a similar way as their equivalent ARIA attributes in HTML.

- **$arialabel**
  the text for the aria label, which is used when a text label is not visible on the form. If there is a label for the control, use the $arialabelledby property instead
- **$arialabelledby**
  the name of a control to act as a label for this control; for example, you could enter the name of a label object to link it to the control. A value in $arialabelledby will override the value in $arialabel; you can use a comma separated list of controls to assign multiple controls as labels for the component
- **$ariadescribedby**
  the name of a control used to describe this control: similar to $arialabelledby, but could be used to provide more information or a longer description about the control; you can use a comma separated list of controls to assign multiple controls as labels

In versions prior to Studio 11, $arialabelledby and $ariadescribedby took a space separated list of controls. When converting libraries from 10.2 or earlier, or importing a library from JSON, spaces in $arialabelledby and $ariadescribedby will be replaced with commas.

You should note that JavaScript controls now have an $active property which works alongside $enabled allowing you to make controls active, inactive, enabled, or disabled, which helps you control accessibility and tab order in your remote forms.

Image Based Controls

You can assign an Alt text value to image-based controls, such as Picture and Activity, using the $alttext property:

- **$alttext**
  a short text to describe the appearance or function of an image, and equivalent to the "alt" attribute in HTML; this property is relevant for controls that contain an image or have a significant visual appearance, such as the Picture and Activity controls.

Page Panes and Landmarks

So-called "Landmark Roles" in standard accessibility guidelines allow you to identify different areas of a form to allow screen readers to describe the structure of the page to end users. You can define Landmarks in your JS remote forms using *Page panes* and by assigning the appropriate value to the $landmark property for each pane: the options for $landmark correspond to the same keywords used for landmarks in the accessibility guidelines (Main, Navigation, Banner).

- **$landmark**
  specifies a role to make the page pane an ARIA landmark region, a kLandmark... constant with kLandmarkNone as the default.

The Landmark options are:

Landmark option	Description
kLandmarkMain	A "Main" landmark which identifies the primary content of the remote form
kLandmarkNavigation	A "Navigation" landmark which identifies an area containing navigation type control or list of links used for navigation
kLandmarkBanner	A "Banner" landmark which identifies an area usually at the top of the form, possibly containing logo, company or application name and search box
kLandmarkContentinfo	A "Contentinfo" landmark which typically identifies common information at the bottom of a form
kLandmarkComplementary	A "Complementary" landmark which many contain supplementary information or further links, such as a sidebar
kLandmarkForm	A "Form" landmark which identifies an area containing a number of input controls or other form controls
kLandmarkSearch	A "Search" landmark which typically would contain a Search field and button
kLandmarkNone	No landmark definition

Label controls

You can link a Label control to a specific Edit control, or you can tag a label as one of the HTML header types, using the following properties:

- **$labelfor**
  links a label to a control. If you use this with some controls such as the Edit control, the linked control will get the focus if the label is clicked. It can be used in addition to $arialabelledby.
- **$tagtype**
  can be used to set a label’s HTML tag type to one of the header types (<h1> etc.) which would allow the end user to navigate to different sections of a form: the default value is kJSLabelTypeLabel, which is a standard untagged label, and the other values include H1 to H6 for the header types.

Control text

If a control has some text assigned (e.g. a button), the screen reader will read out the text by default, therefore it is not always necessary to assign the ARIA properties to describe such controls. For example, the text for a Button control will be read by the screen reader, if no ARIA properties are specified, however the value in $text will be overridden if you specify $arialabel or $arialabelledby.

Content tips

The Edit control has the $::contenttip property which is a text string which is displayed in the edit field when it is empty and before the end user has entered any text. This can be used in addition to the ARIA label properties, to help label the edit controls on your forms: note it is good practice to add labels to all the edit controls on your form to help with accessibility, so do not rely solely on content tips to describe edit controls.

Keyboard Accessibility

As well as the ARIA properties, the behavior when using various keys to navigate a remote form, or inside more complex controls, has been improved. For example, when the end user presses the Tab key, the focus will jump from one control to another in a remote (web) form, or for complex items such as a Tab bar, the Tab key will put the focus inside the control and the arrow keys can be used to move from one element to another. In addition, the Arrow keys can be used to interact with controls, such as dropdown menus, while Enter and Spacebar can be used to select options or items. The Page Up/Down keys can be used to scroll a form or long list which has the focus.

The following table summarizes what keys end users can use in which JS remote form controls:

Key	Action	Applies to which controls
Tab	Tab to next field (previous with Shift + Tab)	All controls. More complex controls such as datagrid and complex grid will tab t field within the control while editing
Arrow Keys	Move within list a control	List based controls such as List, Droplist, Combo box, Datagrid and Datepickers
Page Up Page Down	Move by multiple lines up or down	List based controls such as List, Droplist, Combo box, Datagrid, Complex Grid
Home End	Move to the start or end of a list	List based controls. Complex grid requires Ctrl to be pressed
Ctrl + [Ctrl +]	Move to next/previous in more complex controls	Complex grid: move to next/previous line Subform sets: move to next/previous s set

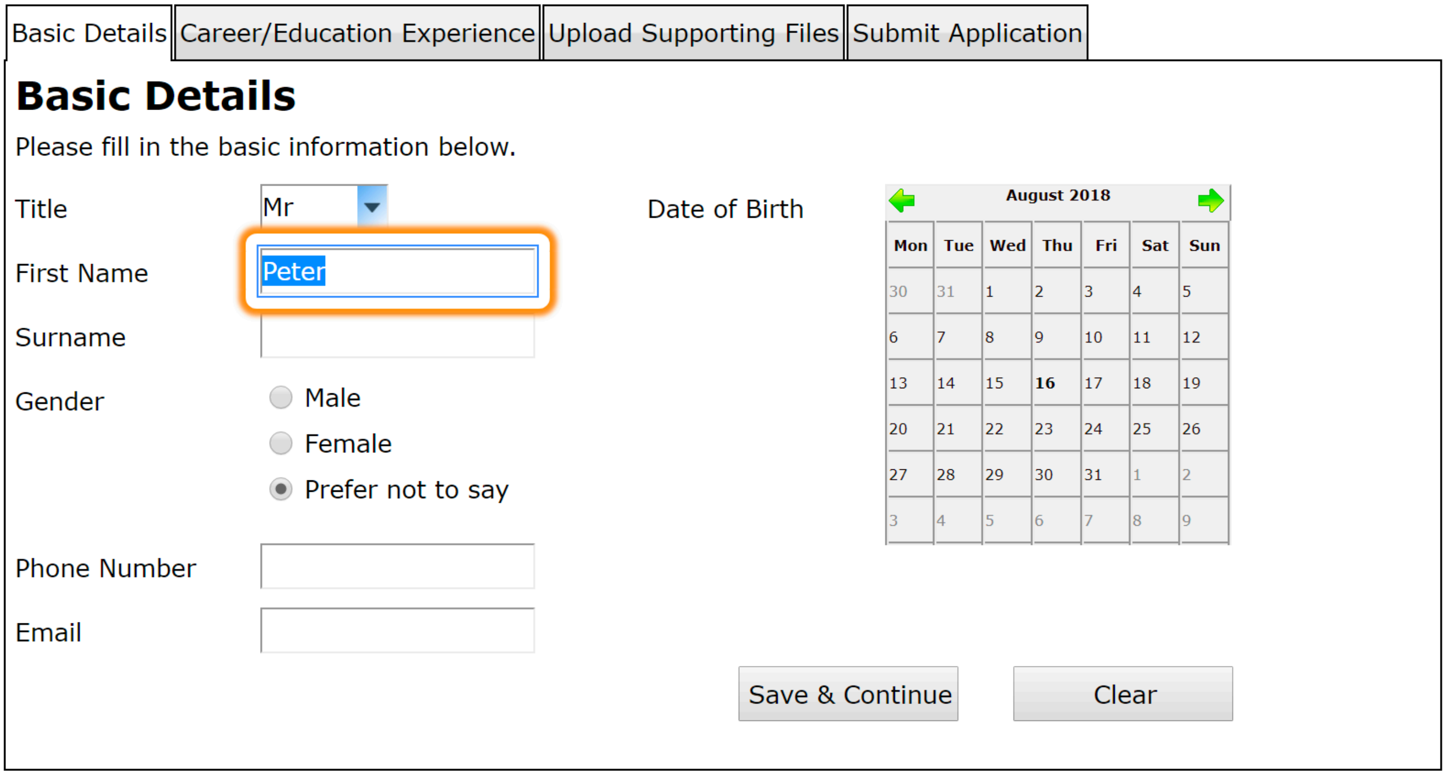
Accessibility and tabbing order

For increased accessibility in your applications, you should carefully consider the tab order of the controls in your forms. In general, it is good practice to make the tab order run consecutively, that is, from one control to the next in a logical order: this could be from left to right starting at the top of the form, but the exact order may depend on the specific functions of your app. The tabbing order of the controls in the form is also used by the screen reader to “read out” or describe the contents of the form, so it’s important how you specify the tabbing order of the fields in your form.

Form Example

With the ARIA labels specified and the correct tabbing order defined, the end user can navigate the controls on a form from the keyboard, and, in addition, the screen reader can describe each control or area of the form page in turn.

Consider the following JavaScript remote form. In the first image, as the end user tabs to the **First Name** edit field, the field border will highlight, the screen reader will say aloud: “First name, Edit text”, and if there is a value in the field, as in this case, it will read that as well: “First name, Peter, Edit text”.



Basic Details

Please fill in the basic information below.

Title: Mr

First Name: Peter

Surname:

Gender: ○ Male ○ Female ● Prefer not to say

Phone Number:

Email:

Date of Birth

August 2018

Mon	Tue	Wed	Thu	Fri	Sat	Sun
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	**16**	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

Figure 115:

Using the Tab key, the end user can move from one control or area of the form to another. Successive tab presses will enter the **Tab bar** at the top of the form, then the Right and Left Arrow keys can be used to move along the Tab bar, and the Return key can be used to select a tab. Once the tab is selected, the screen reader will describe the item selected: “Careers / Education Experience, Tab selected, 2 of 4”.

JS Themes

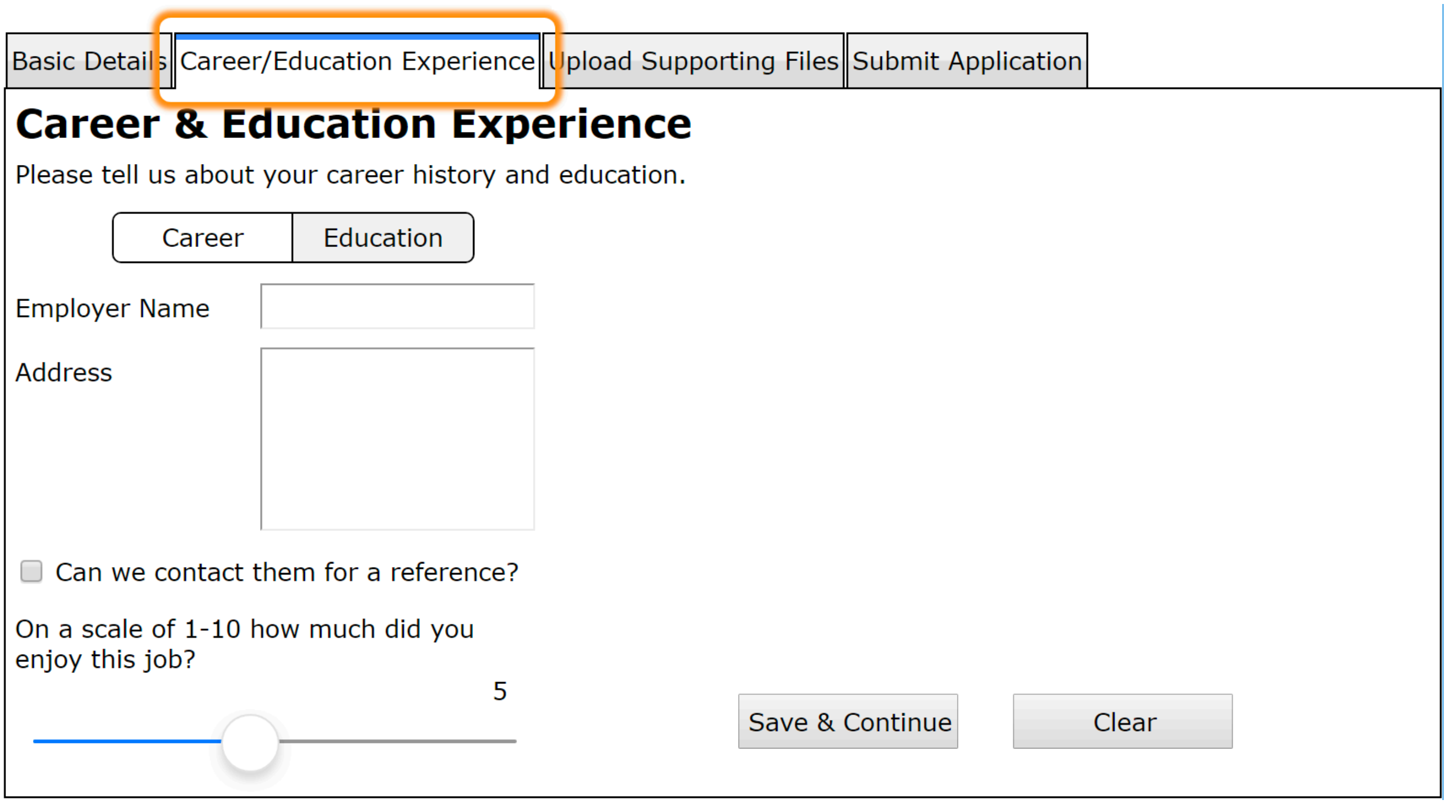
You can apply a consistent set of colors to JavaScript components on a remote form by selecting colors defined in a *theme* – underlying a theme is a set of CSS styles which are applied to controls at runtime on the client. Omnis has a number themes which you can use to style your JS client applications: a **default** theme, which provides an effective and pleasing UI across all JS controls and devices, and a range of different color themes, such as the **dark** theme, which provides an alternative set of darker colors.

When designing a remote form, you can change the current theme in the JS Theme Select dialog by pressing Ctrl-J/Cmnd-J, or select the **JavaScript Theme** option from the **View** menu (you can edit a theme from here by Right-clicking on a theme or background of the dialog and selecting the Open JavaScript Theme Editor option). To select a theme, click on the theme preview and close the dialog. The selected theme is applied to the current remote form and to all the remote forms in your library since the theme is an Omnis-wide preference. The following screenshot shows a Remote form with the ‘vintage’ theme selected.

The current theme is stored in the Omnis root preference, **$javascripttheme** (in $root.$prefs), which is set to the *default* theme initially, and controls which theme is used to render themed colors for all remote forms in design mode (but you can set or change the theme on the client using the ‘settheme’ client command; see later).

Selecting Colors

When you select the color for a JS control, you can choose a theme color from the color palette in the Property Manager, under the Theme color button (the default, on the left) in the color picker toolbar. For example, select a button, click on the Text tab in the Property Manager and click on the color palette for $textcolor.



Basic Details | Career/Education Experience | Upload Supporting Files | Submit Application

Career & Education Experience

Please tell us about your career history and education.

Career | Education

Employer Name

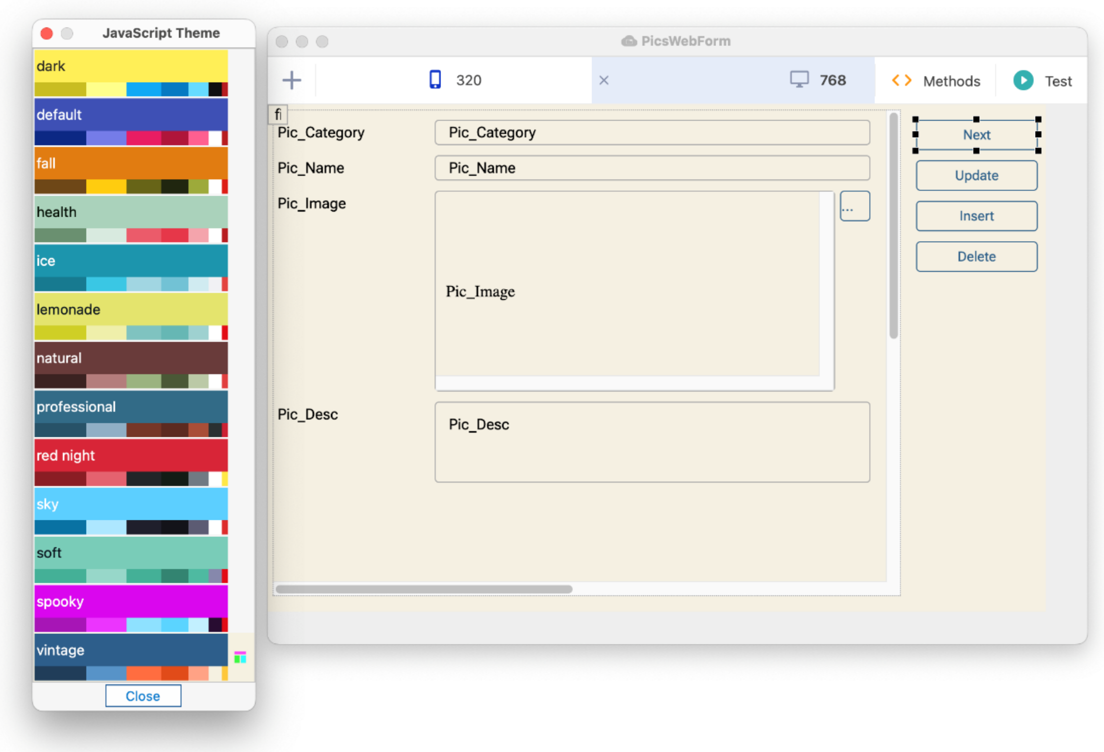
Address

□ Can we contact them for a reference?

On a scale of 1-10 how much did you enjoy this job?

5

Figure 116:



Pic_Category: Pic_Category

Pic_Name: Pic_Name

Pic_Image: Pic_Image

Pic_Desc: Pic_Desc

Figure 117:

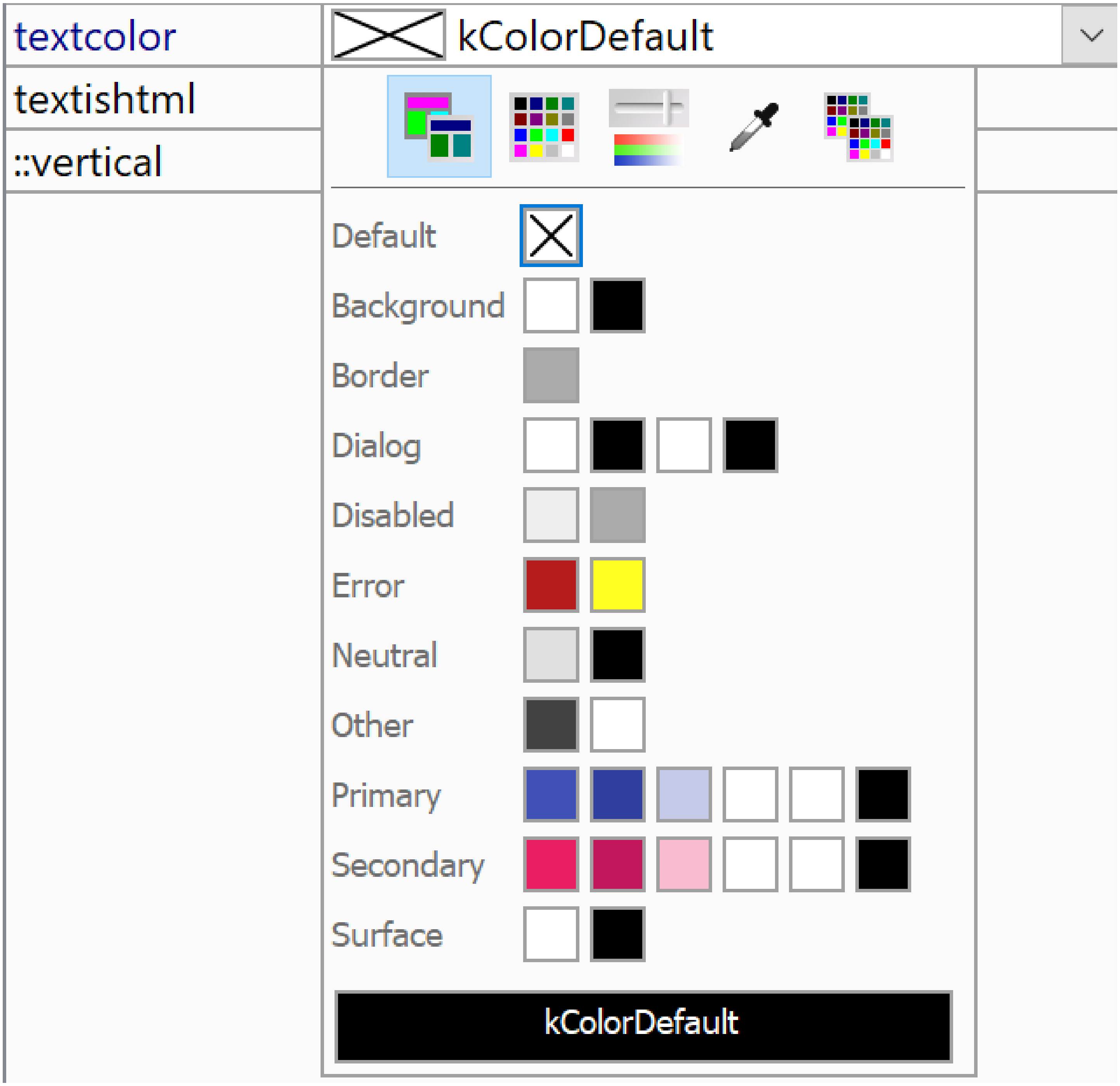

textcolor ☒ kColorDefault

Figure 118:

The color setting for most properties, such as $textcolor, is set to **kColorDefault,** which means the appropriate color from the current theme is used. If a text color property is set to kColorDefault, and it sits on an element with a background color which comes from a themed color constant, the text will be rendered in the associated <theme color>Text color. For example, if a button's $buttoncolor is set to kJSThemeColorPrimary and its $textcolor is set to kColorDefault, the text will be rendered using kJSThemeColorPrimaryText.

The colors defined in a theme and shown on the color palette have corresponding color constants, whose names begin **kJSTheme-Color,** as follows:

kJSThemeColorBackground
kJSThemeColorBackgroundText
kJSThemeColorBorder
kJSThemeColorDialog
kJSThemeColorDialogText
kJSThemeColorDialogTitle
kJSThemeColorDialogTitleText
kJSThemeColorDisabled
kJSThemeColorDisabledText
kJSThemeColorError
kJSThemeColorErrorText
kJSThemeColorFocusedRow
kJSThemeColorFocusedRowText
kJSThemeColorNeutral
kJSThemeColorNeutralText
kJSThemeColorPrimary
kJSThemeColorPrimaryDark
kJSThemeColorPrimaryDarkText
kJSThemeColorPrimaryLight
kJSThemeColorPrimaryLightText
kJSThemeColorPrimaryText
kJSThemeColorSecondary
kJSThemeColorSecondaryDark
kJSThemeColorSecondaryDarkText
kJSThemeColorSecondaryLight
kJSThemeColorSecondaryLightText
kJSThemeColorSecondaryText
kJSThemeColorSurface
kJSThemeColorSurfaceText

Theme Editor

You can create JS themes, or modify an existing theme using the JS Theme Editor, available under the **Add-Ons > Web Client Tools** menu option and select **JavaScript Theme Editor.**

The editor provides a preview of the current theme on the right side of the editor screen, and you can click on an area or text item within the preview to view or set its color (you can also set colors by clicking in the list on the left).

The colors in a theme are categorized as **Primary** and **Secondary,** plus there are specific color for **errors, borders, dialogs,** and so on. The primary colors are used throughout your application and set the general tone or style of the theme, while the secondary colors provide an accent to certain parts of the UI.

Creating a new theme

To create a new theme, *you can duplicate an existing theme* and make any changes to the copy. To do this, open the Theme Editor, select a theme from the dropdown list or use the *default* theme (selected initially by default), click on **Save as** and give the new theme a name – then change individual colors and use the **Save** option to save any modifications. The **Set theme** option sets the $javascripttheme preference to the theme currently shown in the editor. If you make any modifications to the current theme, all open remote forms will be updated automatically.

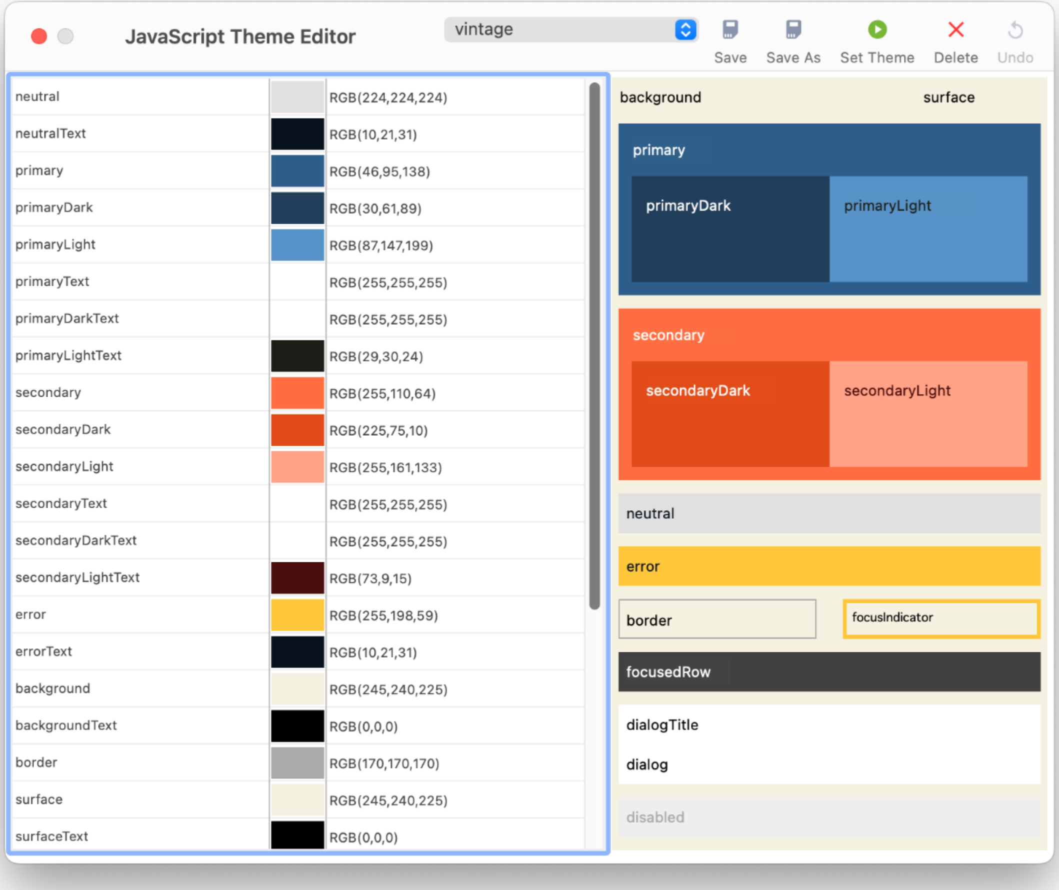
A theme is stored as a **.json** file and an associated **.css** file in the 'html/themes' folder. When deploying your application, the themes folder and its contents must be copied to the corresponding location on the Omnis App Server.

When designing the colors in a new theme, you may want to follow the guidance provided by the Google Material design system, which may help you create a theme containing colors which complement one another and provide maximum useability and accessibility across different platforms and devices. Google provides a Material Color Tool which you may find useful to create a set of complementary colors for the dark/light variants.

Themed Icons

Omnis supports the use of SVG images for component icons. SVG icons can be "themed" which means an icon will be tinted using the control's text color as specified in the current JS theme (the 'fill' color in a themed SVG file is set to the text color from the theme). This allows *a single themed SVG icon file* to be used with different themes and its color is set automatically.

Omnis includes an icon set named 'material' which contains over a 100 themed SVG icons. The material icon set is located in the 'iconsets' folder and if you have used any of the icons in your app the icon set needs to be copied to the Omnis App Server when deploying your application.



neutral	RGB(224,224,224)
neutralText	RGB(10,21,31)
primary	RGB(46,95,138)
primaryDark	RGB(30,61,89)
primaryLight	RGB(87,147,199)
primaryText	RGB(255,255,255)
primaryDarkText	RGB(255,255,255)
primaryLightText	RGB(29,30,24)
secondary	RGB(255,110,64)
secondaryDark	RGB(225,75,10)
secondaryLight	RGB(255,161,133)
secondaryText	RGB(255,255,255)
secondaryDarkText	RGB(255,255,255)
secondaryLightText	RGB(73,9,15)
error	RGB(255,198,59)
errorText	RGB(10,21,31)
background	RGB(245,240,225)
backgroundText	RGB(0,0,0)
border	RGB(170,170,170)
surface	RGB(245,240,225)
surfaceText	RGB(0,0,0)

Figure 119:

The following are examples of a single icon from the material icon set with different color themes applied (note the icon is rendered using the button text color):

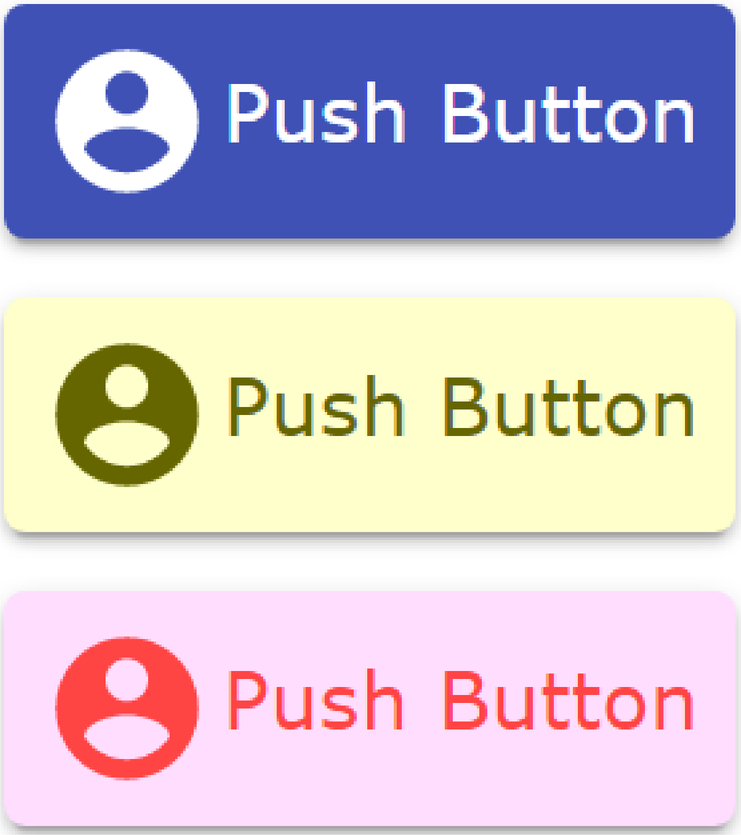


Figure 120:

SVG icon files can be ‘themed’ using the **SVG Themer** tool under the **Tools >> Add Ons** menu option. You can open a single SVG file, preview it using one of the test colors (note the preview colors are not saved to the file), and save it using the **Export** button.

The SVG Themer tool converts a standard SVG image file into an Omnis themed SVG file format: specifically, the first element in the root svg element in the original file is converted to a ‘g element’ with fill=“var(–om-tint-color)” and id ‘omTheme’ which reference the color from the current theme. The Image Data tab shows the source for the converted SVG file which you can edit if required, although the converter should convert the SVG file as necessary.

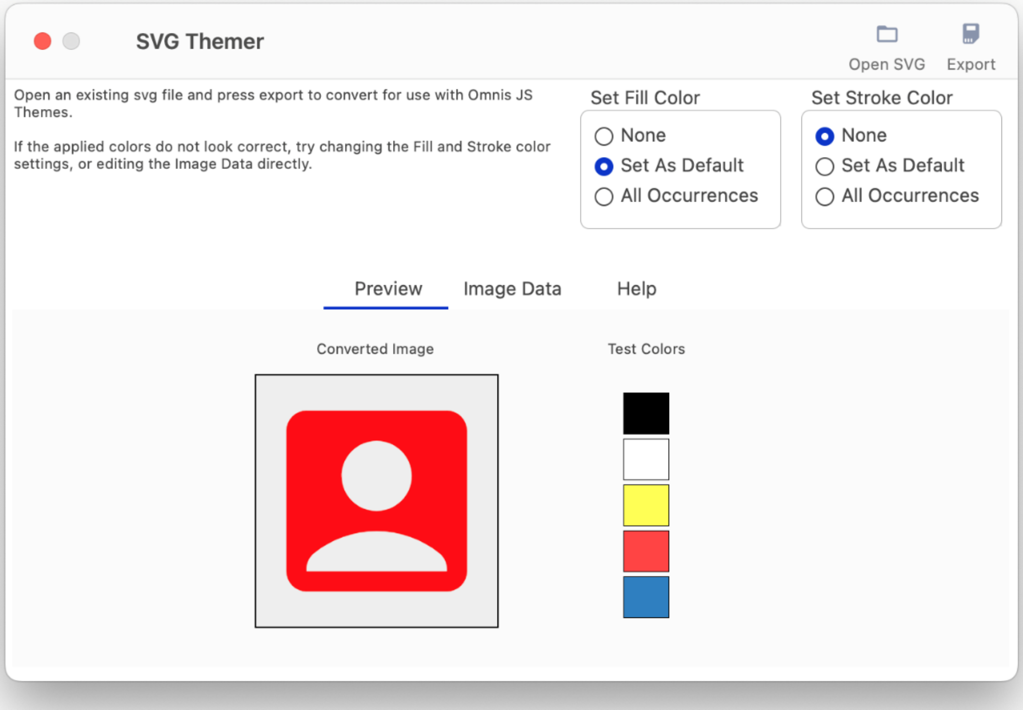
Like other SVG icon files, any themed SVG icons need to be placed in an icon set folder. For example, you could create or acquire a set of SVG icons and convert them using the SVG Themer tool ready for use in your JS client apps. In order to use your own SVG iconset(s), you need to add the icon set name to the $iconsets library property in your library, which can contain one or more icon set names separated by commas.

HTML Template & Client theme setting

The JS client’s theme is set in the ‘data-themename’ attribute in the omnisobject div in the HTML file for your remote form, e.g. data-themename=“dark”.

The special value of “_JT_” is used in the HTML template (jsctempl.htm) which is replaced this with the current value of $javascript-theme when Omnis generates the HTML file for your remote form.

In addition, the ‘data-appid’ attribute specifies the application a page belongs to. It defaults to ‘<lib name>.<form name>’ each time a form is tested (the ‘_APPID_’ placeholder in the template .htm file is replaced when a from is tested).



Open an existing svg file and press export to convert for use with Omnis JS Themes.

If the applied colors do not look correct, try changing the Fill and Stroke color settings, or editing the Image Data directly.

Set Fill Color

○ None
● Set As Default
○ All Occurrences

Set Stroke Color

● None
○ Set As Default
○ All Occurrences

Figure 121:

The current theme: $construct

The current theme is passed in the $construct row parameter, in a column named *theme*.

Changing the Theme

You can change the theme on the JS client in your code using the 'settheme' client command ($clientcommand) which takes a row parameter whose first column is the name of the new theme. Note that a remote form needs to be reloaded in the browser for a change of theme to take effect. Once you have set the theme using the 'settheme' clientcommand, the client stores it in the client localStorage and will use that theme for subsequent visits to the page. To revert to the default theme specified in the HTML page, you need to call the 'settheme' clientcommand, passing an empty string as the theme name (or clear the client's localStorage).

Passing the theme by URL

You can specify a theme when opening a remote form in a browser by passing an extra theme parameter. You can pass the 'omTheme' URL query parameter when loading a remote form (HTML page) in a browser to specify the JS theme to use for the form (and all other forms in the application during that browser session). For example, to specify the dark theme, use the following URL:

```
http://127.0.0.1:9110/jschtml/jsForm.htm?omTheme=dark
```

Active and Enabled Properties

All JavaScript controls have the *$active* property, which is set to kTrue for all new controls (except the Label Control which is kFalse). The $active property allows you to control whether a component is *active* (kTrue) or *inactive* (kFalse) – in an inactive state, a component *cannot be interacted with at all,* so the end user cannot tab to it, the contents cannot be selected or scrolled (in a list), and user clicks on an inactive control are ignored. Therefore, when a control is inactive, it is completely ignored in the tabbing order, so when the end user tabs the focus will jump to the next active control – in the context of accessibility, an inactive component will be ignored.

The $enabled property allows you to disable a control, and many controls have this property, including the Bar Chart, Combo Box, Data Grid, Date Picker, Edit, List, Map, Pie Chart, Rich Text, and Tree list.

Default Inactive Appearance

When controls are inactive, that is, when $active=kFalse, they tend to have their own default inactive appearance, which is often a gray overlay or background. If you set $defaultinactiveappearance to kFalse you can override this default inactive appearance. The default value for $defaultinactiveappearance is kTrue to maintain backwards compatibility.

Default Disabled Appearance

All controls that have the $enabled property have the $defaultdisabledappearance property. When $enabled = kFalse the $defaultdisabledappearance property defaults to true and the 'omnis-notenabled' css class is applied to the client element of the control. If $defaultdisabledappearance = kFalse, this class is not applied, which is what sets the text colour to grey when disabled.

Context Menus

Context menus are opened if $active of the control is kTrue, regardless of $enabled. If you wish to disable this behavior for a control, you should use *Quit event handler (discard event)* when handling evOpenContextMenu in your event handling methods for the control.

Creating Customized JavaScript Components

You can add your own customized JavaScript components to the Component Store under your own tab. This might be useful if you always want to create edit controls or buttons with certain properties (e.g. colors or fonts), so creating your own custom JS components might save you some time. You will need to edit the Component Store library (comps.lbs) to change the contents of the Component Store.

To create your own customized JS components, you need to create a new JS remote form in the Component Library, copy any components you want to customize from the JSFormComponents form, add them to your own form, and then customize then as required.

Opening the Component Library

You can open the Component Library by Right-clicking on the **Libraries** node in the Studio Browser tree and selecting the **Show Comps.lbs** option. Unless you want to copy classes from another library to the Component Library, it is recommended that you close all other libraries before you edit the Component Library.

When you select the Show Comps option, the contents of the Component Library is displayed in the Browser Browser. You can switch the Browser to Details view and click on one of the column headings to show the different types of component either by Name or Type. You can hide and show specific types of class in the Component Library using the Class Filter option in the Browser.

Adding your own form and components

We recommend that you do not change the components in the JSFormComponents form since these are the default components that appear in the Component Store, rather you should create your own customized components using the following method.

Add a new JS remote form class to the Component Store library (comps.lbs); note that the name of the new remote form will be used as the tab name in the Component Store toolbar so choose something appropriate. Set the $componenttype property of the remote form to kCompStoreDesignObjects using the Notation Inspector: to do this, open the Notation Inspector, click on the Search button (the cursor changes to a spy glass), click on your remote form in the Studio Browser, and in the Property Manager set $componenttype to kCompStoreDesignObjects (note the $componenttype property will only be displayed via the Notation Inspector). Then set the $layouttype of the new remote form to kLayoutTypeSingle (not kLayoutTypeResponsive).

With your new remote form open, open the JSFormComponents remote form next to it (this form contains all JS components). Drag any JavaScript controls you want to customize from JSFormComponents into your new remote form and change their properties or appearance as required. After you hide the Component Store library, the customized JavaScript controls will be available in the new group in the Component Store.

JavaScript Component Templates

When you add a JavaScript Component to a remote form *in your code at runtime,* Omnis uses a template to create the object with all the required properties and methods. There is a template for every type of JavaScript Component, and the templates are located in the \studio\componenttemplates folder.

The component templates match the default components in the Component Store, and should not be edited. There are templates for report and window class components as well.

Position Assistance

When you *move or resize objects* in the Remote form design editor (or window class editor), colored dashed lines and arrows will appear automatically that enable you to easily align and distribute controls and other objects in relation to the design window edges or center and any other nearby objects – this is called **Position Assistance.**

As you move or resize objects on a remote form, *colored lines* are shown automatically, and objects will snap into position to help you arrange the objects in a form. In addition, the *position* and *size coordinates* are shown directly under the object or group of objects. For example, when you place several fields on a form, Position Assistance can help you align their left-hand or top edges and ensure they are spaced evenly. Position Assistance is also provided when you use the Arrow keys to position or resize objects.

The **Show Position Assistance** option on the remote form context menu allows you to toggle the Position Assistance feature (enabled by default). There is a single setting for this, shared by all editors, that is saved to omnis.cfg when Omnis shuts down.

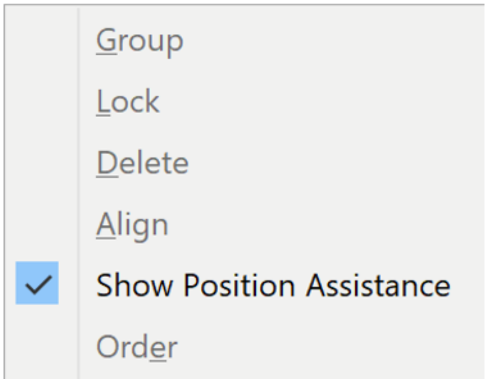


Figure 122:

The positioning lines are drawn using the **colorhighlight** color in the system group of appearance.json. The entry positionAssistantKeyboardTimer in the 'ide' section of the config.json specifies the time that the position assistance remains visible after you stop pressing an arrow key; this defaults to 750 milliseconds.

When positioning objects in the center of a remote form, Position Assistance uses the center of *the current layout breakpoint*, not the center of the remote form design window.

Position and Size coordinates

When you *move* or *resize* an object, or group of objects in a remote form class, the *position* and *size* information for the object or group is shown automatically. In addition, *position* information is provided when you drag an object from the Component Store and drop it onto a remote form.

The current ***Position*** of an object (its x,y coordinates) is displayed in a helptip or colored box just below the object, when you **move** an object, or when you drag an object from the Component Store and drop it onto a remote form (see above left); the helptip shows the X,Y position of the top-left corner of the object relative to the top-left corner of the remote form (or window class design screen).

The current ***Size*** of an object is shown (width x height) when you **resize** it (see above right). When more than one object is selected, the position or size corresponds to the area of the whole group of selected objects.

The **positionAssistantShowsPositionOrSize** item in the 'ide' section of the config.json file allows you to enable or disable this feature (the default is true, so the position or size is shown).

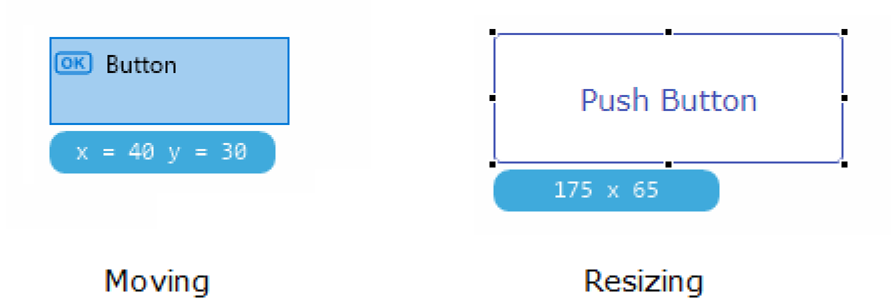


Figure 123:

Positioning & Aligning Objects

When the Position Assistance is enabled, Omnis gives precedence to distribution over alignment, and within alignment it prioritises the top edge, over the center, and the center over the right edge. As soon as a visual guide is displayed for a target, any other targets that would also cause the object to move in the same axis are dropped.

As you move or size objects Omnis displays a visual guide when the object(s) being moved or sized are within +/-2 pixels of a specific alignment or distribution target, e.g. an alignment target is the top edge of another object or objects. When you release the mouse, the objects will snap to the displayed target. Position Assistance is applied to objects dragged from the Component Store, as well as objects being moved or sized within a design window. Position Assistance is provided when moving an object even if the adjacent objects are contained inside a container field.

When sizing objects, assistance is not provided if the objects being sized have more than a single container, that is, the component that is the parent of the objects – this can be more granular than a field, such as for complex grids, there are several containers such as the row and header sections.

Position Assistance is provided within each section of a Complex Grid, that is, the row and header sections of a Complex Grid, and the above behavior for container fields applies to each section independently.

Distribution

Position Assistance attempts to *distribute objects* by allowing them to be evenly spaced. The visual guide for distribution is a line drawn between the objects with arrow heads.

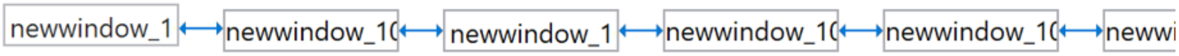
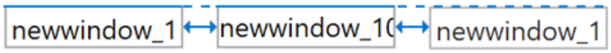


Figure 124:

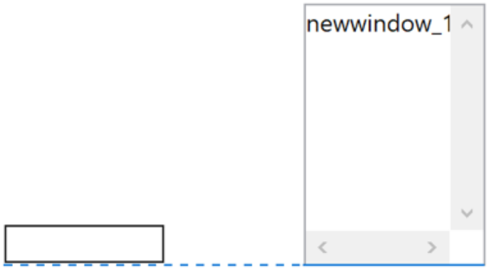
The guides are drawn for as many objects as possible, immediately adjacent to the object(s) being moved or sized. Position Assistance works best when objects are already reasonably well arranged, either vertically or horizontally, so for more complex arrangements, with overlapping fields may result in no visual guides being presented.

Alignment

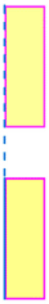
Position Assistance attempts to *align objects* by giving them the same *top* or *bottom* coordinate, or *centered* relative to each other. When you try to center objects, you only get visual guides when moving objects, and when the appropriate side of the rectangle representing the objects being moved either fully encloses or is fully enclosed by the appropriate side of the object in which it is being centered. The following illustrate how the Position Assistance is applied for different cases when aligning objects.



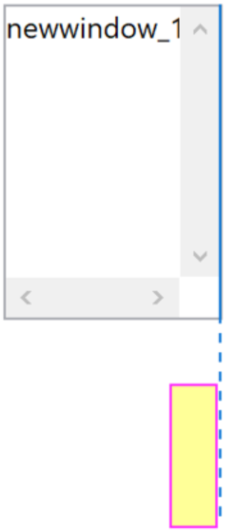
Top alignment



Bottom alignment



Left alignment



Right alignment

Positioning for Paged Panes (Container fields)

Assistance is provided to help you align fields inside a container field, such as a Paged Pane. In addition to the left/right, top/bottom positioning, when you move an object inside and *near to the center of a container*, a line across either the vertical or horizontal center of the container is drawn and the object will snap to the line.

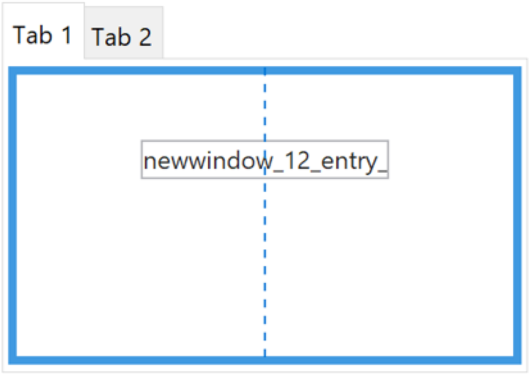


Figure 125:

When positioning objects *inside a Paged Pane* (or any container), Position Assistance is only provided for the controls *within the Paged Pane* itself, so objects outside the Paged Pane are not included in the current object grouping. Similarly, if you are positioning objects *outside, but near to* a Paged Pane, the objects inside the Paged Pane are not included in the current grouping.

Position for dropping objects

Information about the drop destination is shown when dragging objects on a remote form design window *into a Complex grid or Paged pane* (shown in addition to the x-y coordinates). For example, as you drop a field into a Complex grid section, the position assistance will display the section type, such as 'Header', 'Horizontal header', or 'Row', as shown below.

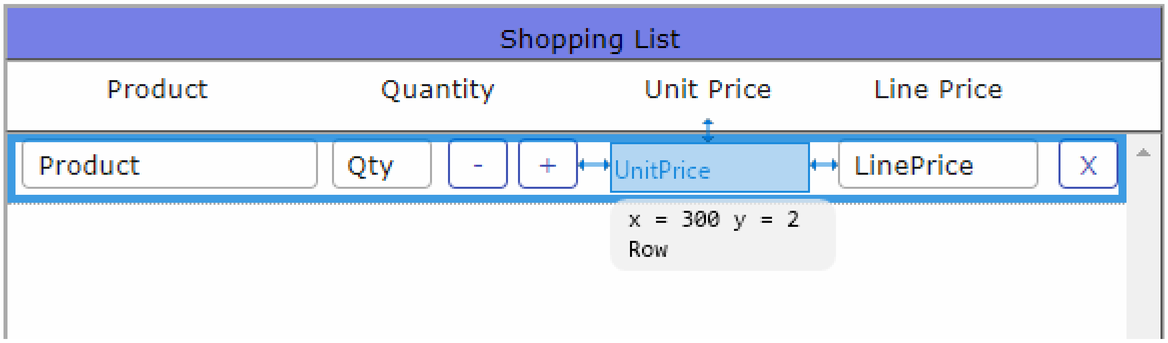


Figure 126:

Group Selection & Object Properties

When you select a group of objects, Omnis shows a colored line around the group and *a single set of selection handles* for the group (this also applies to objects in a window or report class). If the objects share any properties these are shown in the Property Manager allowing you to set the properties for all the objects in the group.

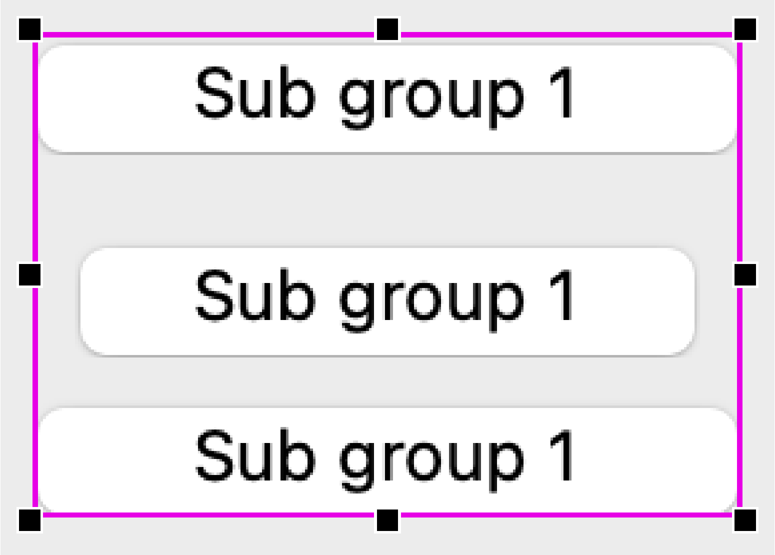


Figure 127:

If you click on an object inside the group of selected objects, Omnis shows selection handles around just this object and shows the properties of the selected object in the Property Manager, but retains the selection line around the group, as shown below as a gray line:

You can click on another object within the group selection, and in this case selection handles are shown on the new selected object and its properties are shown in the Property Manager. If you want to restore the state where the properties reflect all the selected objects in the group, you can Shift-click on the currently selected object.

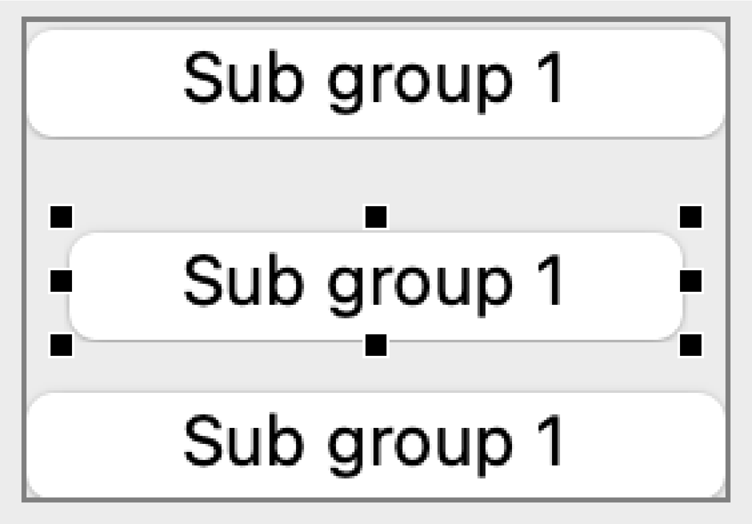


Figure 128:

Note that when clicking, properties are not shown until you release the mouse. This allows you to drag the selected objects without changing the properties displayed in the Property Manager.

The color of the selection rectangle shown around a group of objects is one of two colors in the 'IDEGeneral' section of appearance.json:

- **designselectedgroupoutlinecolor**
  the color of the rectangle around a selected group (when no single object is selected)
- **designselectedgroupoutlinesinglecolor**
  the color of the rectangle around a selected group when a single object is selected inside the group

Activity Control

Group	Icon	Name	Description
Other		Activity	Animated image to display during process or Omnis Server activity

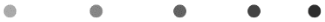
The **Activity Control** provides an animated image to show some activity on the client, for example, during a long list calculation or search operation on the Omnis Server.

There is an example app called **JS Activity** in the **Samples** section in the **Hub** in the Studio Browser showing the different activity controls available and how to set a custom animated Gif; the same app is available in the JavaScript Component Gallery.

The Activity Control has the following custom properties:

Property	Description
$activitystyle	The style of the indicator, a constant as below, or kJSActivityCustomLink to specify your own image
$customlink	The path of an animated GIF which can be displayed when $activitystyle is set to kJSActivityCustomLink

The $activitystyle property specifies the style of the control, one of the following constants:

Constant	Description
kJSActivityBar	
kJSActivityBlock	
kJSActivityCircular	
kJSActivityLinear	
kJSActivityLinearDots	
kJSActivitySmallSpinner	
kJSActivityCustomLink	the image in $customlink is used

Custom Link

The following code assigns a custom link to the activity control:

```
Calculate iCustomLink as 'http://www.mywebsite.com/images/animated1.gif'\
Do $cinst.$objs.Activity.$customlink.$assign(iCustomLink)
```

As an alternative to using the Activity Control you could consider using the *showloadingoverlay* client command to show a loading indicator (animated image) over a remote form or specific control: see Custom Loading Indicator.

Background Shape

Group	Icon	Name	Description
Shapes		Background Shape	Object you can set to Rectangle, Li Triangle, or Image

The **Background Shape** allows you to draw various shapes in your remote forms: you assign the shape to the object by setting the $::shape property to one of the kJSBack... constants. It can be assigned one of a number of shapes including: Ellipse/Circle, Rectangle/Square, Rounded rectangle/Square, Triangle, Horizontal Line, Vertical Line, and Image.

You can assign a solid color or gradient fill to a background component by setting its $backpattern, $forecolor and $backcolor. You can also assign the stroke (border) thickness and color by setting $strokewidth and $bordercolor.

There is a sample app called **JS Background** in the JavaScript Component Gallery, and under the **Samples** option in the **Hub** in the Studio Browser showing the various background shapes.

Background Images

You can create a background image by setting $::shape to kJSBackImage, and setting $imagepath to a URL relative to the Omnis tree during development or your web server for deployment; you can click on the $imagepath property to select an image.

You can drop an image from your system / desktop on to a Remote Form to create a background image. In this case, the $imagepath property is set to the path of the image, which is copied automatically to the folder 'images/libs/<libname>' in the html folder in the Omnis tree. The image can be a PNG, JPG, JPEG or SVG.

The $keepaspectratio property is set to kTrue by default which ensures the image will keep its aspect ratio.

Animations and Changing Attributes

The Background Control has the $animation and $attr properties which allow various animations or effects to be assigned to the object, such as fading the object in or out, or for $attr various attributes of the object to be changed. The $animation and $attr properties must be assigned at runtime and accept a string containing various parameters depending on the function or attribute.

There is an example app called **JS Animations** in the **Samples** section in the **Hub** in the Studio Browser showing how can use animations to move and fade objects to create a richer UI; the same app is available in the JavaScript Component Gallery.

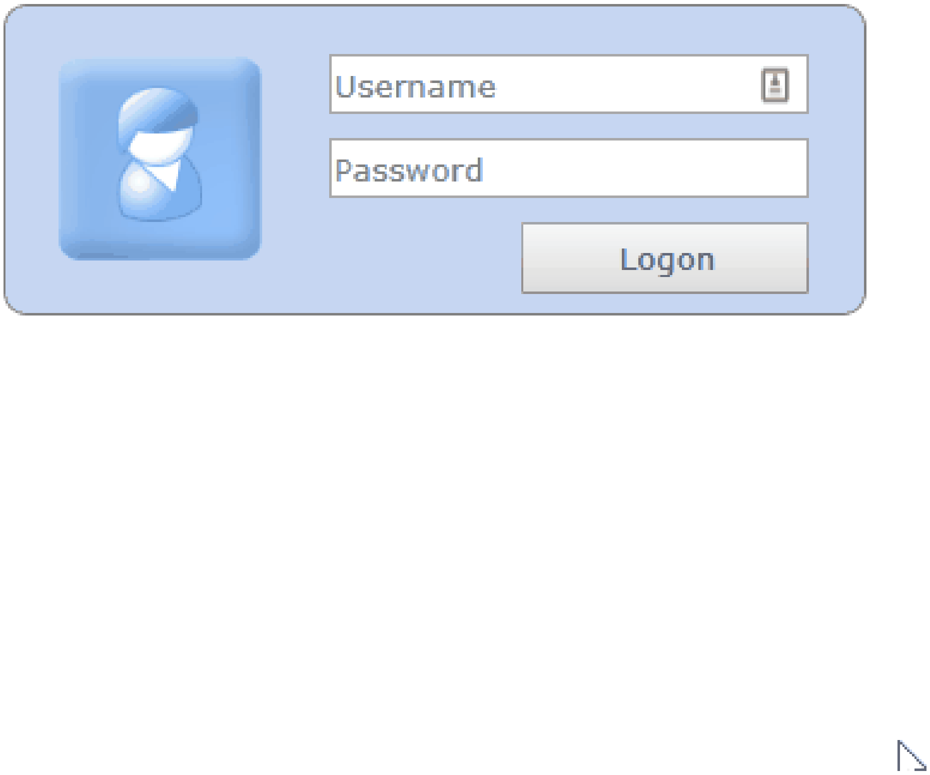


Figure 129:

Apart from the scale attribute, the browser must support the Raphael JavaScript library to allow animations and attribute changes (more details about the parameters you can use are available from http://raphaeljs.com).

The $animation property follows the general format:

```
function( newvalue, time(milliseconds), ease(optional), complete_context(optional) )
```

The functions available in $animation are:

- **scale**
  increases and decreases the size of the object

```
scale( newvalue, time(milliseconds), ease(optional), complete_context(optional) )
```

- **alpha**
  changes the transparency of the object

```
 alpha( newvalue, time(milliseconds), ease(optional) , complete_context(optional) )
```

- **rotate**
  rotates the object

```
rotate( newvalue, time(milliseconds), ease(optional) , complete_context(optional) )
```

For example:

```
Calculate $cinst.$objs.backgroundobject.$animation as "alpha(0,500,<>,fade_complete)"
```

fades the object to alpha value 0 over 500 milliseconds using a 'slow, faster, slow' easing method (see below) and when complete calls evAnimComplete with a parameter "complete_context".

You can use the complete event to chain the next animation, therefore to pulse an object you could use:

```
Calculate $cinst.$objs.backgroundobject.$animation as "alpha(0,500,<>,fade_off)"
```

You could use the event handling method:

```
On evAnimComplete
  If ( pAnimContext="fade_off" )
    Calculate $cwind.$objs.backgroundobject.$animation as "alpha(0,500,<>,fade_on)"
  Else if ( pAnimContext="fade\_on" )
    Calculate $cwind.$objs.backgroundobject.$animation as "alpha(0,500,<>,fade_off)"
End If
```

Ease transition effects

The following ease transition effects or "eases" are supported for animations:

Transition	Description
=	linear and default if not specified
>	fast then slowing
<	slow then faster
<>	slow, faster, slow
bounce	object bounces
elastic	object stretches
backIn	object backs in
backOut	object backs out

Changing Background Attributes

The $attr property allows you to change various attributes of the object, such as their transparency. For example, you can assign an alpha gradient to an object using the following method:

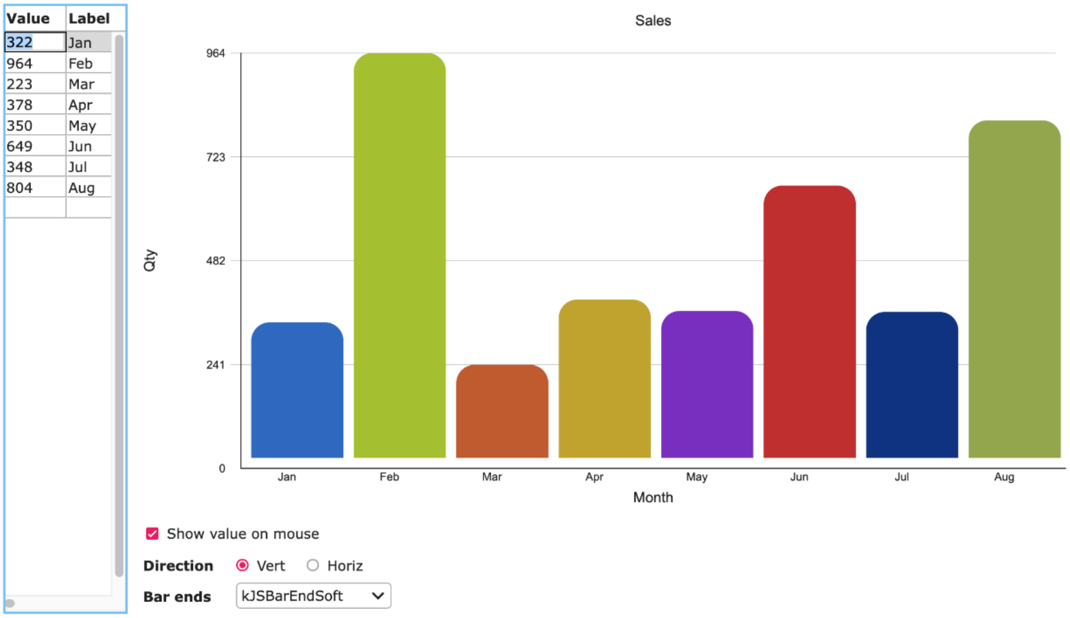
```
 # note must be assigned at runtime
 Do $cinst.$objs.backgroundobject.$attr.$assign('attr(gradient, 0-\#FFFFFF:10-\#FFFFFF)')
 Do $cinst.$objs.backgroundobject.$attr.$assign('attr(opacity,0.0)')
```

Bar Chart Control

Group	Icon	Name	Description
Visualization		Bar Chart	Displays a bar chart based on a list

The **Bar Chart** control allows you to display a simple dataset contained in a list variable as a bar chart. Omnis Studio provides two chart controls: a **Bar Chart** control and a **Pie Chart** control. The data to be represented in both these controls is contained in a *list variable* which is assigned to the $dataname property of the control.

There is a sample app for both the **JS Bar chart** and **JS Pie chart** in the JavaScript Component Gallery, and under the **Samples** option in the **Hub** in the Studio Browser.



Value	Label
322	Jan
964	Feb
223	Mar
378	Apr
350	May
649	Jun
348	Jul
804	Aug

☑ Show value on mouse

Direction ● Vert ○ Horiz

Bar ends kJSBarEndSoft

Figure 130:

There are various properties (on the Appearance tab in the Property Manager) that allow you to control the appearance of a Bar or Pie chart. The bars and segments use a set of default colors, but you can specify your own colors at runtime.

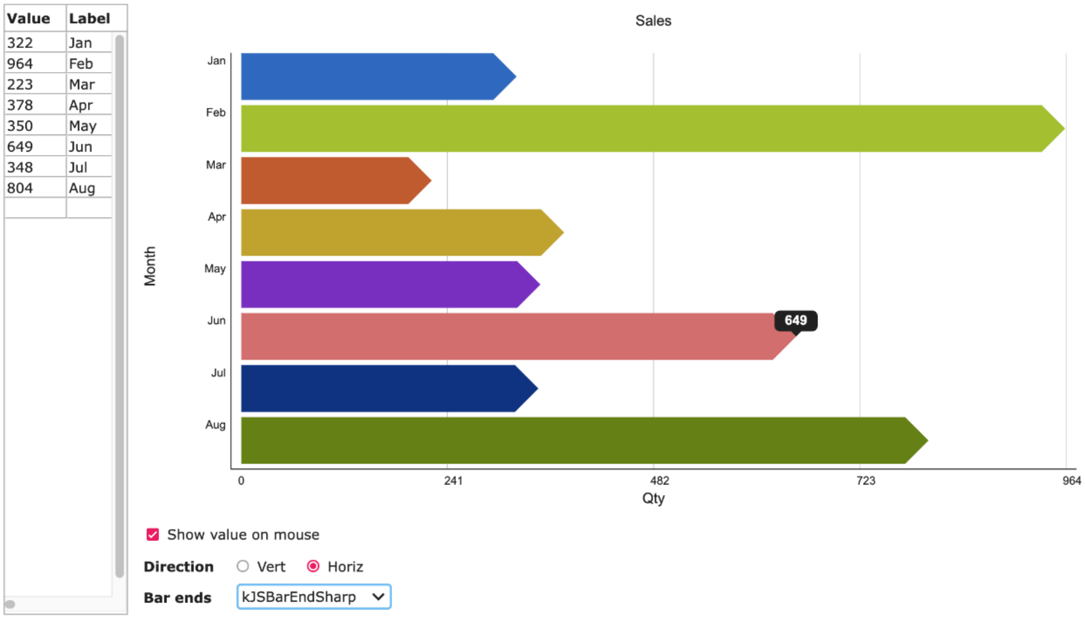
For Bar charts you can set $chartdirection to vertical (the default) or horizontal bars, the style of the bar ends as the $barends property (kJSBarEndSharp is shown below), and you can display data values when the end user's mouse hovers over the bar by setting $showvalue to True (the default).

Specific properties for Pie charts are described under the PieChart Control section.

List data structure

To draw a simple bar or pie chart, the list variable assigned to the $dataname property of the chart component needs to contain at least two columns. The first column contains the **value** for the data point, and the second column contains the **label** or name for the data point. For example, to construct a simple bar chart showing a list of figures for sales agents, you could use the following method:

```
# create vars bar_data (List), amount (Number), name (Char)
Do bar_data.$define(amount,name)
Do bar_data.$add(120,'Steve')
Do bar_data.$add(230,'Dave')
Do bar_data.$add(245,'Anita')
Do bar_data.$add(125,'Claire')
Do bar_data.$add(280,'Ben')
```



Value	Label
322	Jan
964	Feb
223	Mar
378	Apr
350	May
649	Jun
348	Jul
804	Aug

☑ Show value on mouse

Direction ○ Vert ● Horiz

Bar ends kJSBarEndSharp

Figure 131:

With the $showvalue property enabled (a value is displayed when the end user passes the pointer over each bar), the method produces the following chart.

When $showvalue=kTrue, the popup label will use $backcolor for the text and $textcolor for the background of the label so that it can be seen against the background of the control.

Main and Axis Titles

There are a number of properties in the Bar Chart to allow you to add a main title, as well as titles for the x and y axis. In addition, there are properties to show (the default) or hide the x and y axis details or units.

Property	Description
$maintitle	The main title for the chart
$xtitle	title for the x axis
$ytitle	title for the y axis
$showxaxis	if kTrue the chart shows x-axis details
$showyaxis	if kTrue the chart shows y-axis details

The following chart shows all the titles and axis details.

Text and Axis Colors

The $textcolor and $axiscolor properties allow you to set the color for text and axis, including theme colors; $textcolor applies to the color of the title, labels, axis text, and legend in the chart, where applicable.

The $axiscolor property for a Bar chart applies to the color of the both axes lines, and the unit lines which run across the bar chart.

When set to kColorDefault, both properties will set their color dynamically according to the color of $backcolor.

Bar & Segment Color

You can specify your own colors for the bars or segments in a Bar or Pie chart using the runtime-only property $colorlist, rather than using the default colors. You need to create a list of strings representing CSS colors and assign the list to the $colorlist property, for example:

```
Do iColorList.$define(iColor)
Do iColorList.$add("\#CE3D3D")
Do iColorList.$add("rgb(81, 206, 61)")
Do iColorList.$add("hsl(230, 60%, 52%)")
Do iColorList.$add("Gold")
Do $cinst.$objs.PieChart.$colorlist.$assign(iColorList)
```

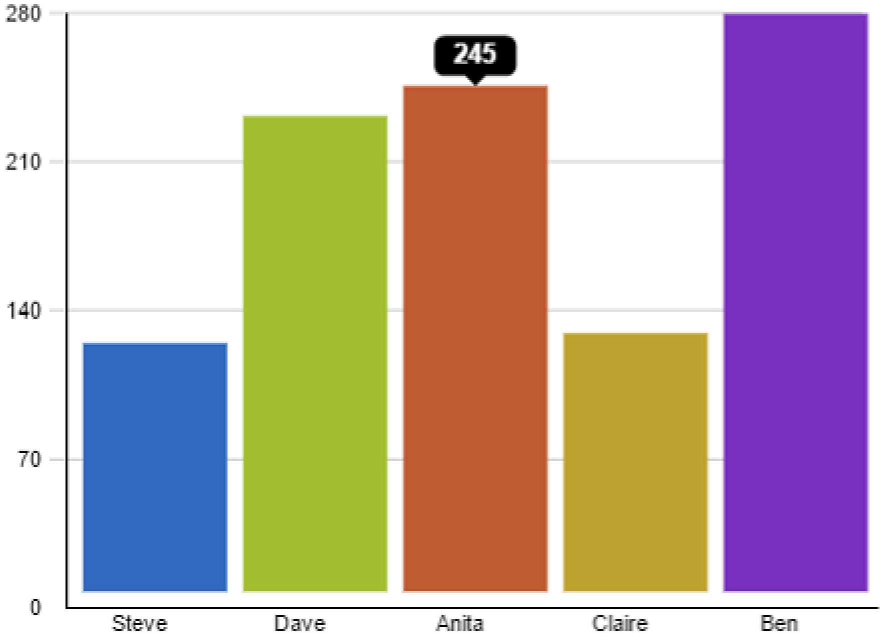


Figure 132:

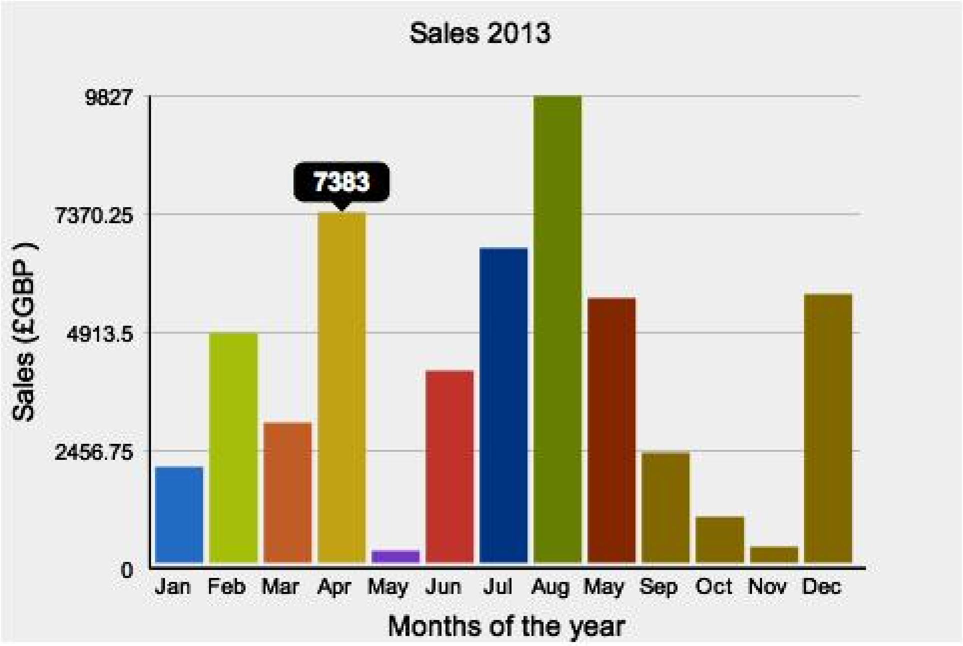


Figure 133:

Events

Bar charts report the evBarClicked event with the bar clicked in the pBar parameter. Similarly, Pie charts report the evSegmentClicked event with the segment clicked reported in pPieSegment.

```
# event method for bar chart, message is a field on the form
On evBarClicked
  Calculate message as con("Bar clicked: ",pBar)

# event method for pie chart
On evSegmentClicked
  Calculate message as con("Segment clicked: ",pPieSegment)
```

Button Control

Group	Icon	Name	Description
Buttons	OK	Button	Standard pushbutton which reacts

The **Button** control is a basic pushbutton that the end user can click with the pointer or tap on a mobile device to confirm something or initiate a process, such as an OK or Cancel button, or a Print or Send button. Alternatively, you can use a Split button which combines a button and droplist of preset options. Many of the sample apps in the JavaScript Component Gallery contain buttons, including plain text buttons or ones with icons.

Apply
Print

Figure 134:

The text is specified in **$text** as a single line of plain text, unless **$textishtml** is set to true and the text is treated as HTML; see below. A button can display an icon, specified in the **$iconid** property (under the Appearance tab), which can be an SVG or PNG image file from an Icon set chosen from the Select Icon dialog. See earlier in this chapter about Component Icons and specifically SVG Icons.

In addition to the icon you can assign to a button, you can add 'Icon Badges' to button icons to provide additional information, such as a number count, a notification, or an alert: see Icon Badges.

The Button control has the following text and appearance properties:

Property	Description
$align	The alignment of the text inside the button; centered by default
$buttonbackiconid	The icon id of background image for the button. To use the default system button, set $buttonbackiconid to zero and $buttoncolor to kColorDefault

Property	Description
$buttoncolor	The color of the button. To use the default system button, set $buttonbackiconid to zero and $buttoncolor to kColorDefault
$borderwidth	The border width in pixels (the default is 0 or no border)
$bordercolor	Sets the border color when $borderwidth is set >0
$textbeforeicon	If true, and the control has both text and an icon, and the text is displayed to the left of the icon
$::vertical	If true, the text and icon are arranged vertically
$textishtml	Specifies that the text entered in $text is treated as HTML
$isflat	If true, the button has a flat appearance

Text Position

In addition to $align, there are several layout properties that give you greater control over the positioning of the text and icon on the button (also applies to the text for the **Trans** button and **Split button** components):

Property	Description
$vertalign	The vertical alignment or justification of the text and icon within the button, a constant: kJstVertTop, kJstVertMiddle and kJstVertBottom
$vertpadding	The top and bottom padding of the text and icon within the button (default is 4 pixels); only applies when $vertalign is kJstVertTop or kJstVertBottom
$spliticonandtext	If true, the icon and text are separated so that the text can be aligned independently (default is kFalse)
$icontextspacing	The gap between the icon and the text when they are positioned together (default is 4 pixels)

When $spliticonandtext is kTrue, the icon is positioned at the edge of the button (on the left by default). The text can be aligned in the remaining space with the $align or $vertalign property.

You can enter negative values for the properties requiring a number of pixels, which may be required in some circumstances.

You can use the existing $::vertical property to arrange the icon and text vertically, and $textbeforeicon to display the text before the icon; after setting these to kTrue, you can use the align and padding properties to position the text.

Flat Button Style

The flat style of a button is controlled using the $isflat property. The style for all new buttons (and buttons in converted libraries) is flat (from Studio 10.2 onwards), so $isflat is set to true. In addition, if the value of $buttonborderradius in converted libraries is set to 0, it will now be changed to the new default of 4; any other value will be retained on conversion.

When $isflat is disabled a button has a small drop shadow and when the button has the focus a larger difused drop shadow is displayed around the button.

Disabled Appearance

The appearance for flat buttons when they are disabled ($active = kfalse) is as follows: if $isflat is kTrue (the default), the button back color will become transparent (if it isn't already) and the text color will take on the disabledText color. If $isflat is kFalse, the button back color will take on the disabled color and the button text color will take on the disabledText color.

In addition, if $bordercolor for buttons is set to kColorDefault the color will match $textcolor. When disabled ($active = kFalse), the border will match the disabled text color to maintain a consistent disabled appearance.

HTML Button Text

When set to kTrue the $textishtml property specifies that the text for the button (entered in $text) is treated as HTML, therefore any HTML can be used to style the text. For example, you can insert a line break by setting this property to kTrue, and using <br> in $text for the button wherever a line break is required.

The $textishtml property also allows other styling of the button text using various character and color attributes. Note that design mode does not render the HTML (the raw HTML code is displayed), and if you use attributes in the HTML they must be enclosed in single quotes.

Events

When a Button is clicked an evClick event is triggered which you can handle in the event handling method behind the button.

```
On evClick
  Do something…
```

Example

The sample apps in the **Applets** section in the **Hub** have an About window which is loaded into a subform and displayed using an animation; see the Animations section for the About button code. The **Close** button on the About windows simply closes the About form by sending a message to the main remote form to run a method; it has the following code:

```
On evClick
  Do $cwind.$closeAbout()
```

In this case, $cwind is a reference to the main parent form which contains a method called $closeAbout which contains code to fade out the About form and reset various buttons on the main form.

Camera Control

Group	Icon	Name	Description
Media	📷	Camera	Allows the end user to capture ima scan QR codes or barcodes

The **Camera** control allows the end user to capture images or scan QR codes and barcodes from within your application; this could be the camera on a mobile phone, tablet, or laptop, or a video cam attached to a desktop PC.

You can set the capture mode by setting the $cameraaction property to one of the kJSCameraAction… constants. When returning an image the $dataname property must be set to a Character or Binary type instance variable to receive the image (not required for barcode scanning); for Character variables, the captured image is stored as base64 encoded data.

Camera Actions

The **$cameraaction** property allows you to set the action or mode on the current device for the camera to capture an image, QR code or barcode. $cameraaction is a runtime only property that should be assigned a row with 1 to 3 columns as row(*action* [,*mode, deviceId*]), where *action* is a kJSCameraAction... constant, *mode* is a kJSCameraFacingMode... constant, and *deviceId* is a character string of the device ID.

Constant	Description
kJSCameraActionGetDevices	Gets a list of camera devices attached to the user's device, sent to evGetDevices. Requires only action column, other values will be ignored.
kJSCameraActionStartCamera	Starts the camera and shows viewfinder to prepare to capture an image; note this is not required for scanning codes. Requires at least 2 columns, with column 2 (*mode*) set to one of the following: **kJSCameraFacingModeDeviceId** uses a specific camera on the end user's device, specified by deviceId required in column 3; **kJSCameraFacingModeUser** selects the user facing camera on the device; **kJSCameraFacingModeEnvironment** selects the environment facing camera on the device.
kJSCameraActionCaptureImage	Captures a still image from the camera after kJSCameraActionStartCamera. It is recommended to assign this action in response to a client executed method for best performance and user experience. Resulting image data will be assigned to the variable in $dataname. Requires only *action* column, other values will be ignored.
kJSCameraActionStartBarcodeScanner	Starts the camera in QR code/barcode scanner mode. evBarcodeScanned will be fired upon detection of a code. Requires at least 2 columns, with column 2 (*mode*) set to kJSCameraFacingModeUser.
kJSCameraActionStop	Stops the current camera feed (both in image capture or barcode scanning mode). Requires only *action* column, other values will be ignored.

Camera UI

The **$showui** property allows you to show the appropriate UI for using the Camera, Barcode scanner, and to switch between the front and back camera.

The $showui property takes a kJSCameraUI... constant (or sum of constants) to specify which UIs are shown in the control: **kJSCameraUINone** displays no UI in the control (the default), **kJSCameraUICamera** shows the UI for using the camera (Start Camera, Take photo and Stop camera buttons), **kJSCameraUIBarcode** shows the UI for scanning barcodes (Start and Stop Scanner buttons), and **kJSCameraUISwitchCamera** shows a button to allow the end user to switch between the front and back cameras.

The **$iconid** property can be used to specify an icon to be shown in the control to indicate which mode the camera is in; the icon is not shown when the camera is in video mode. For example, you could show the *photo-camera* icon if the control is in camera mode, or you could show the *qr-code-scanner* icon if the control is in scanner mode; both these icons are available in the material icon set.

Camera Permission and Testing

Use of the camera requires the end user to accept a prompt which is popped up automatically when trying to access the camera for the first time. *This cannot be bypassed,* so if the end user denies access to the device Camera, the actions will not work.

In addition, camera access using a mobile device is only possible when serving over HTTPS. Therefore, you will not be able to access the camera on a mobile device connected to the same network, as Omnis only serves over HTTP for testing. However, you can test a remote form that uses the Camera control locally on your development machine. A utility to serve your localhost server over the internet using HTTPS can be used as a workaround, such as ngrok.

Image Aspect Ratio

If specified, the **$aspectratio** property forces the Camera control to maintain the aspect ratio of the image. You need to specify a number representing the aspect ratio, such as:

$aspectratio value	Description
0	Uses device default
1	A square ratio, 1:1
1.333334	Standard camera ratio, 4:3
1.777778	Wide ratio, 16:9

Providing a non-standard aspect ratio may lead to unexpected results, such as the camera feed not showing at all. Note that the orientation of the camera is set by the device, therefore a desktop/laptop camera will tend to display in landscape orientation, while a mobile camera will show in portrait orientation.

Capture Size

The **$capturesize** property should be an integer and forces the size of the captured image; if empty, the image is captured at the size specified on the device camera. The value specifies the size of the longest edge of the image using the $aspectratio to set the other edge. For example, if a standard ratio of 4:3 is used, and the users device captures at 1024 x 768, a $capturesize value of 640 will produce an image of 640 x 480.

Image Type & Quality

The $imagetype property should be set to a constant to indicate the type of image to be captured. Due to limited support across browsers, only PNG or JPEG (kJSCameraImageTypePNG or kJSCameraImageTypeJPEG) are supported.

If $imagetype is set to JPEG, you can specify a quality level in $imagequality to reduce the data size, on a scale of 0-100 with 100 being the maximum quality.

Events

The Camera control reports the following events:

Event	Description
evGetDevices	Fired in response to kJSCameraActionGetDevices being assigned to $cameraaction. Returns pCameraList, a list containing 2 columns, DeviceId and DeviceDescription. The value in DeviceId can be used for specifying a specific camera to use when starting the camera or barcode scanner
evImageCaptured	Fired when an image has been captured and the instance variable in $dataname has been updated. Returns pImageType, the image data type as an integer

Event	Description
evBarccodeScanned	Fired in response to scanning a valid code, with pValue containing character data of the read code, and pCodeFormat containing a character representation of the code format, e.g. QR_CODE or CODE_128

Due to the web browsers required only to support PNG files we include an integer parameter, pImageType, to state the image data type, in case the selected type was not supported by the browser. In the case that a browser does not support the selected type, it will always use PNG. Most modern browsers support JPEG, which is why we have included JPEG support, but it is best to check for your own use case before using JPEG over PNG.

All JS Camera events have an additional **pError** param which reports any possible errors. The pError parameter is a row containing two columns: the errorCode column will contain a kJSCameraError... constant, and errorDescription will contain the error information from the browser. The error constants are:

Constant	Description
kJSCameraErrorAbort	An Abort error has occurred
kJSCameraErrorNotAllowed	A Not allowed error has occurred
kJSCameraErrorNotFound	A Not found error has occurred
kJSCameraErrorNotReadable	A Not readable error has occurred
kJSCameraErrorOverconstrained	An Over constrained error has occurred
kJSCameraErrorSecurity	A Security error has occurred
kJSCameraErrorType	A Type error has occurred
kJSCameraErrorUnknown	An Unknown error has occurred

Chart Control

Group	Icon	Name	Description
Visualization	📈	Chart	Displays different chart types inclu Line, Bar, Radar, Pie, Doughnut, Po Scatter and Bubble

The **Chart** component allows you to create different types of charts from list data to display in a remote form. It uses the Chart.js JavaScript library, an open source library available under the MIT license, which you can use in your applications (with the correct license attribution). The JS Chart control provides you with a wider range of chart types than the individual Bar chart and Pie chart JS components, and provides a modern interface for displaying charts, with scalable, vector based shapes and animated transitions.

The $charttype property sets the basic chart type, a kJSChartType... constant, and the following types of chart are available.

Chart type	Description
LineBarRadar	Line, Bar, and Radar type charts (or Labelled charts) use a ***label*** (e.g. a month) for the X axis (horizontal), and a ***value*** for the Y axis (vertical).
PieDoughnutPolarArea	Pie, Doughnut, PolarArea charts (or Area type charts) use the same list definition as the labelled charts, but each data point has a different color and its value is represented by ***area.*** With pie charts, the ***angle*** of a segment represents its value (individual values are taken as a percentage of the sum of values in the dataset). Doughnut charts are the same as pies but have an area cutout of the center of the circular chart. PolarArea charts are similar to pie charts, but the ***radius*** of a segment represents its value (in this case, each segment has the same angle).
ScatterBubble	Scatter charts use ***X*** and ***Y values*** to plot points on the chart. Bubble charts use ***X*** and ***Y values*** to plot the position of a data point, with an additional ***R value*** used as the *radius* or size of the bubble, giving a visual indication of the magnitude of the data point.

All chart types can handle multiple datasets, although in practice some chart types are more suited to certain types of data than others. For bar charts, multiple datasets are stacked next to each other, while in most other chart types, multiple datasets are overlaid each other.

There is an example application called **JS Charts** in the **Samples** section of the **Hub** in the Studio Browser demonstrating all the types of chart available. The following image is a **Labelled Bar** chart in the example app:

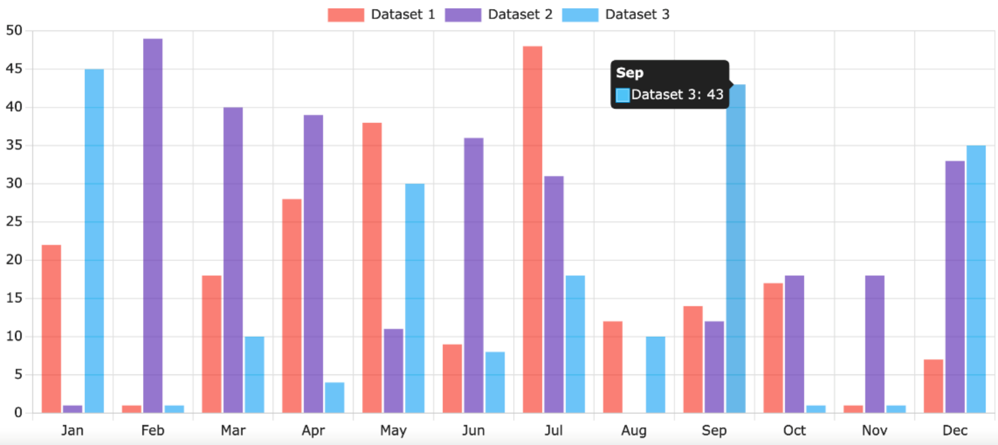


Figure 135:

The following is a **Labelled Line** chart in the example; note the data is displayed in a popup when you pass the pointer over a data point (e.g. Dataset 2 for April is shown).

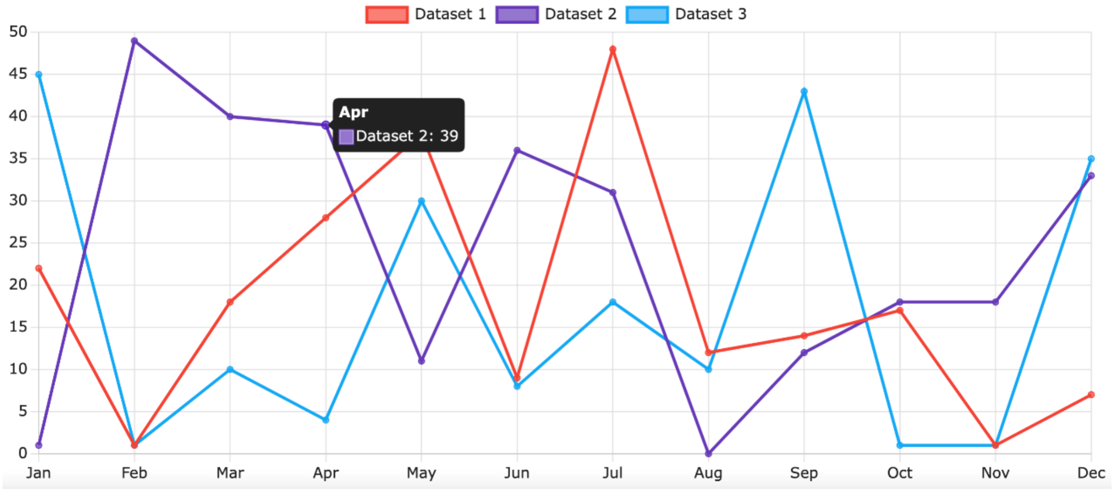


Figure 136:

The following is a **Labelled XYR Bubble** chart; in this case, each data point is plotted using X,Y coordinates and a third value is shown as the Radius (R value) of the bubble indicating the magnitude of the value.

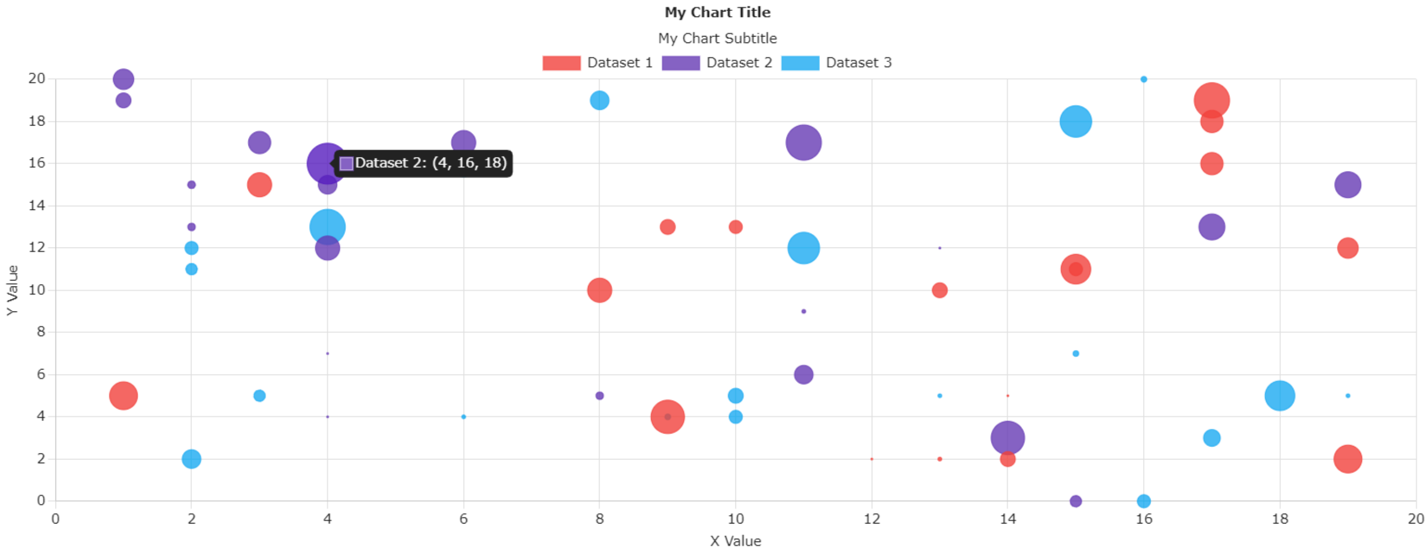


Figure 137:

The following shows two Pie types, a **Doughnut** where values are represented as percentages of the total pie (the same as a pie chart but has an area cutout of the center), and **Polar Area** where the radius (area) of a segment indicates its value.

Chart Data

As with other chart types in Omnis, the Chart control gets its data from an Omnis *list variable,* and the structure or contents of the list needs to match the type of chart you wish to draw. The chart list should contain 2 columns, with *each row in the list representing a dataset:* Column 1 is the data (a list of values for each dataset), and Column 2 is a list of display options relating to that dataset, such as bar or segment colors.

	Data (Col 1)	Options (Col 2)
Dataset line 1	List of Values for dataset 1	Options for dataset 1
Dataset line 2	List of Values for dataset 2	Options for dataset 2
Dataset line 3	List of Values for dataset 3	Options for dataset 3
Etc	…	…

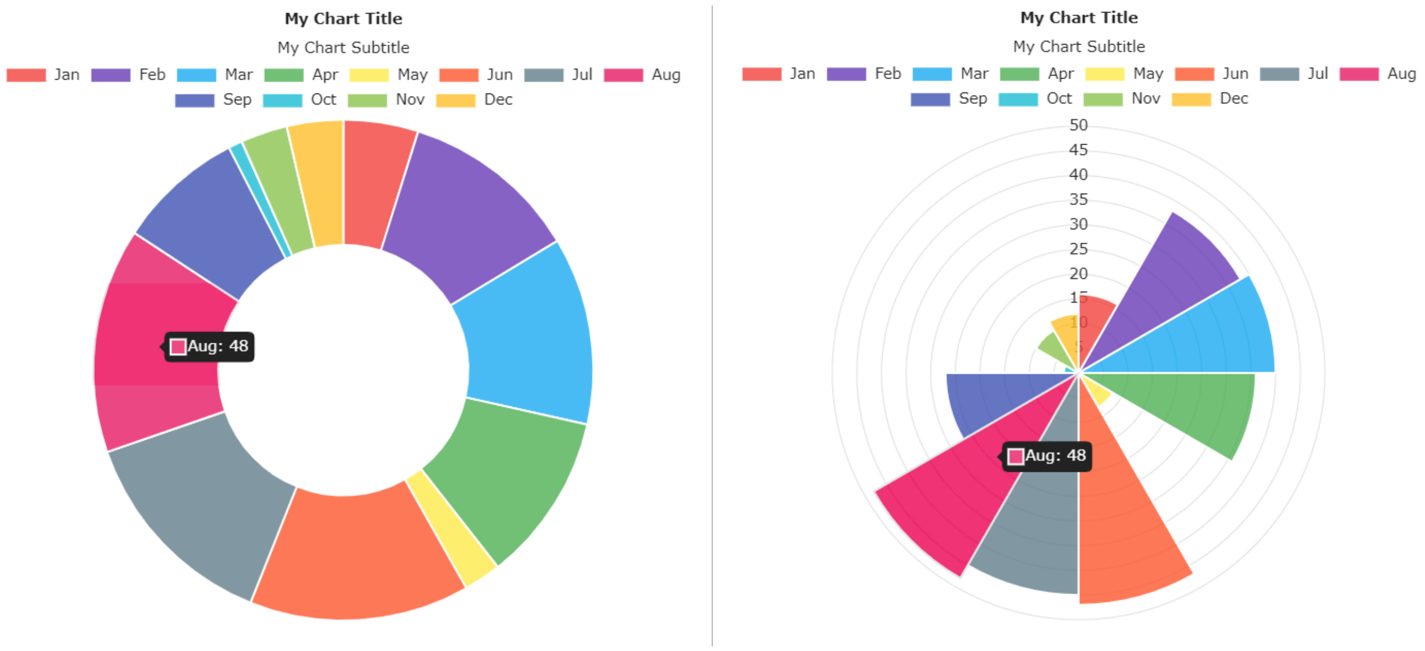


Figure 138:

The ***Data list*** for the chart (in column 1) will vary depending on the chart type as follows:

- The data list variable for *Labelled* and *Area* chart types (e.g. Bar and Pie) requires 2 columns, usually with a Label and a Value:
  Column 1, X axis: Label type data, such as months, exam grades, etc.
  Column 2, Y axis: Value, such as average temperature, number of students, etc.

- The data list variable for *Scatter* (XY) and *Bubble* (XYR) charts requires 2 or 3 columns, respectively, and are in effect points (coordinates) on the chart:
  Column 1, X axis value.
  Column 2, Y axis value.
  Column 3, R value: Bubbles have a Radius, which is given in pixel size.

In the JS Chart example library in the Hub, the chart list for the Labelled Bar chart has the following structure; the main chart list has 2 columns, iData and iOptions. The **Data list** in column 1 has 2 columns, Label and Value (Y), as shown:

The ***Options list*** in Column 2 of the main chart list must be a list of 2 columns containing key-value pairs of Options to apply to that dataset, which are generally display options (colors/rounding on bars/etc).

Looking at the Labelled chart in the JS Chart example library, the Options list in column 2 of the main chart list has the following structure: Key and Value, with entries for **label, backgroundColor,** and **borderColor:**

You can examine the code in the example library to see how the chart data is constructed, for example, look at the $getDatasetOptions class method in jsCharts. The Data and Options data in the example library produces the following chart:

Any options described in the Chart.js documentation should work, however the following are the most useful:

Key	Value	Description
backgroundColor	Valid CSS colors (e.g. #FF0000, rgba(255,0,0,0.5), or theme colors can be used, e.g. kJSThemeColorPrimary. Multiple values can be specified, separated by commas.	Sets the background color of the chart elements in that dataset, i.e. the bars, pie segments etc. If multiple values are supplied these will be applied in order to each element, i.e. 1st bar uses 1st color, 2nd bar uses 2nd color, etc. If there are not enough colors for the data points it will loop back through the given colors.
borderColor	As above	Sets the border color of the chart elements, same as the above.

Key	Value	Description
borderWidth	A number in pixels	Border or line width of the chart elements
borderRadius	A number in pixels	Radius of all corners of the rectangle elements except corners touching the axis or base of chart.
pointStyle	One of: circle, cross, crossRot, dash, line, rect, rectRounded, rectRot, star, triangle	Sets the style of the point in Scatter and Line charts

More options can be found in the Chart.js documentation at: https://www.chartjs.org/docs/latest/charts/. You can look in the Chart Types section to find out which options apply to each chart type, e.g. under 'Styling' https://www.chartjs.org/docs/latest/axes/styling.html.

Any options that can accept arrays of values should be supplied as comma separated values, for example, to have three different background colors you could assign the following line as a value for the backgroundColor key:

```
'rgb(255,0,0),rgb(0,255,0),rgb(0,0,255)'
```

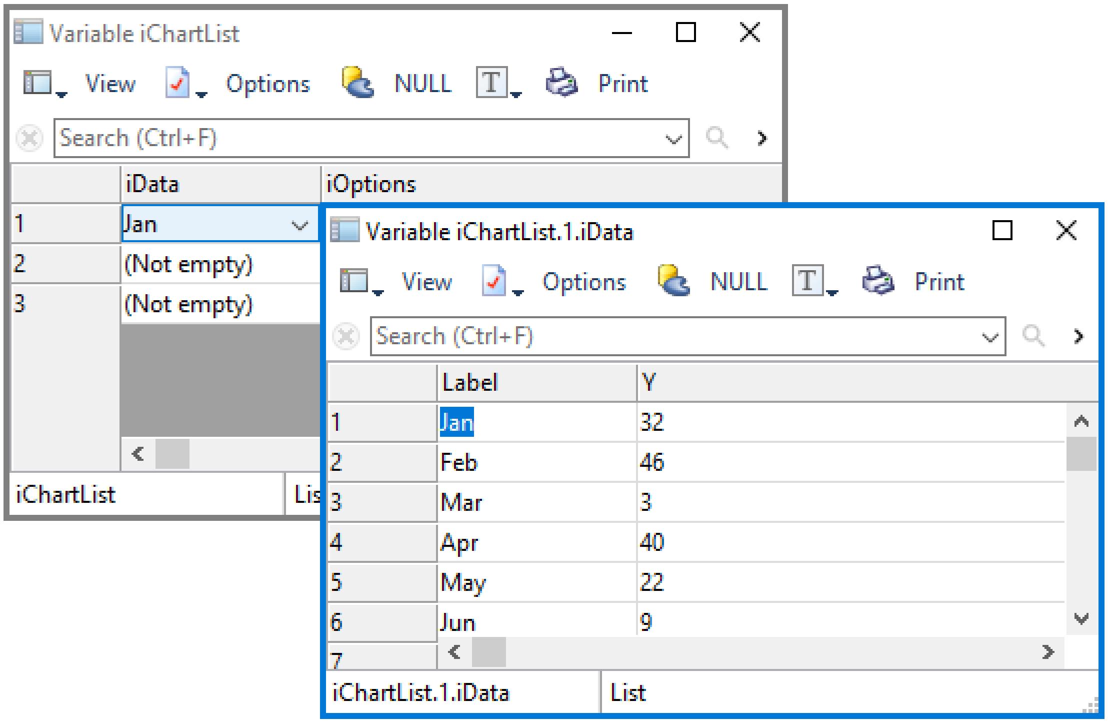
Properties

In addition to controlling the contents of a chart by setting up the list data, you can set various properties for the different chart types. The Chart component has the following properties (some properties may not apply to all chart types).

Property	Description
$dataname	The name of the list instance variable, as described above
$charttype	Sets the basic chart type, a constant: kJSChartTypeLine, kJSChartTypeBar, kJSChartTypeRadar, kJSChartTypePie, kJSChartTypeDoughnut, kJSChartTypePolarArea, kJSChartTypeScatter, kJSChartTypeBubble
$titletext$subtitletext	The title and subtitle text for the chart
$xtitletext$ytitletext	The X and Y title text for Scatter and Bubble (XY) charts
$titleposition$subtitleposition$legendposition	The position of the title, subtitle, and legend, a constant: kJSChartElementPositionTop, kJSChartElementPositionRight, kJSChartElementPositionBottom, kJSChartElementPositionLeft
$legendalign	Aligns the legend element relative to its position, a constant: kJSChartElementAlignStart, kJSChartElementAlignCenter, kJSChartElementAlignEnd
$showlegend	If true, shows the legend
$showdatatooltips	If true, shows tooltips when the pointer is hovered over chart elements
$swapaxes	If true, swaps the X and Y axes; only applies to Bar charts
$disableanimations	If true, prevents the chart from animating
$legendclickhidesdata	If true, the data is hidden from the chart when the user clicks an item in the legend; clicking again will show the data

Events

The Chart control sends the **evClick** and **evLegendClick** events with the following event parameters:



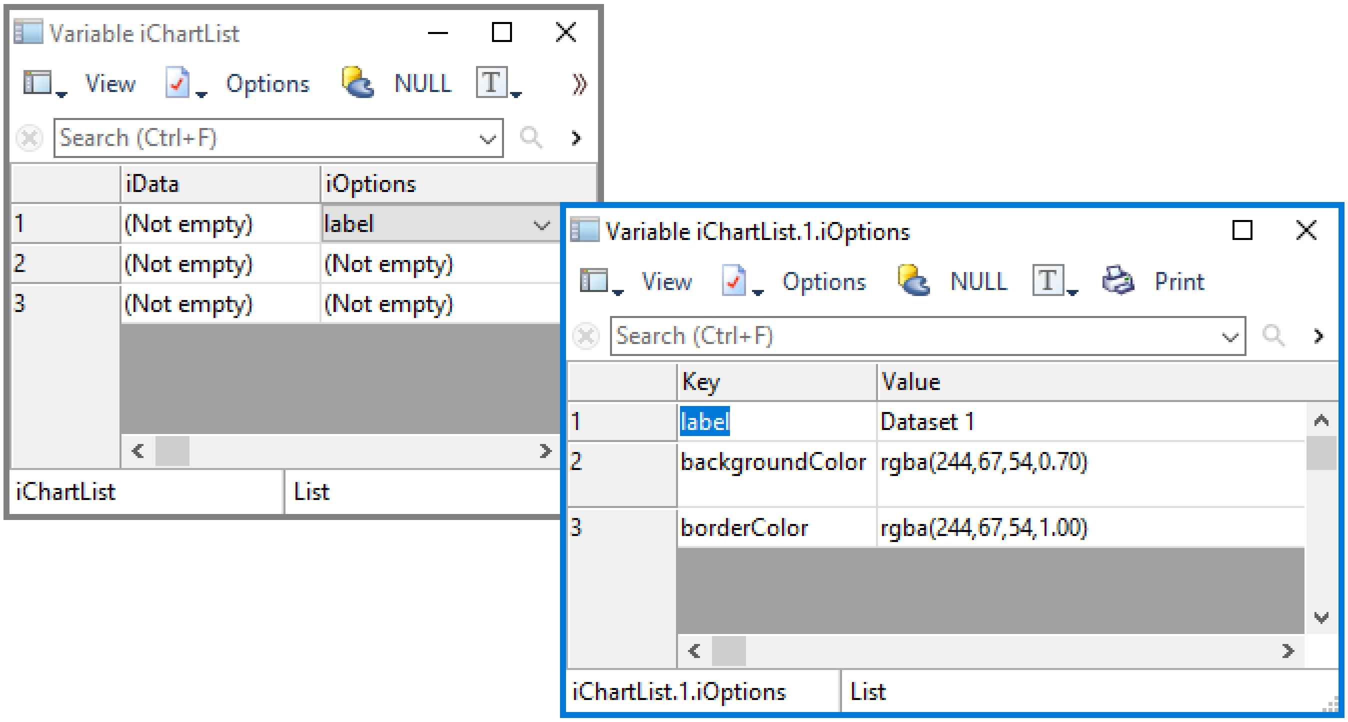
	iData	iOptions
1	Jan	
2	(Not empty)	
3	(Not empty)	

iChartList | Lis

	Label	Y
1	Jan	32
2	Feb	46
3	Mar	3
4	Apr	40
5	May	22
6	Jun	9
7		

iChartList.1.iData | List

Figure 139:



	iData	iOptions
1	(Not empty)	label
2	(Not empty)	(Not empty)
3	(Not empty)	(Not empty)

iChartList | List

	Key	Value
1	label	Dataset 1
2	backgroundColor	rgba(244,67,54,0.70)
3	borderColor	rgba(244,67,54,1.00)

iChartList.1.iOptions | List

Figure 140:

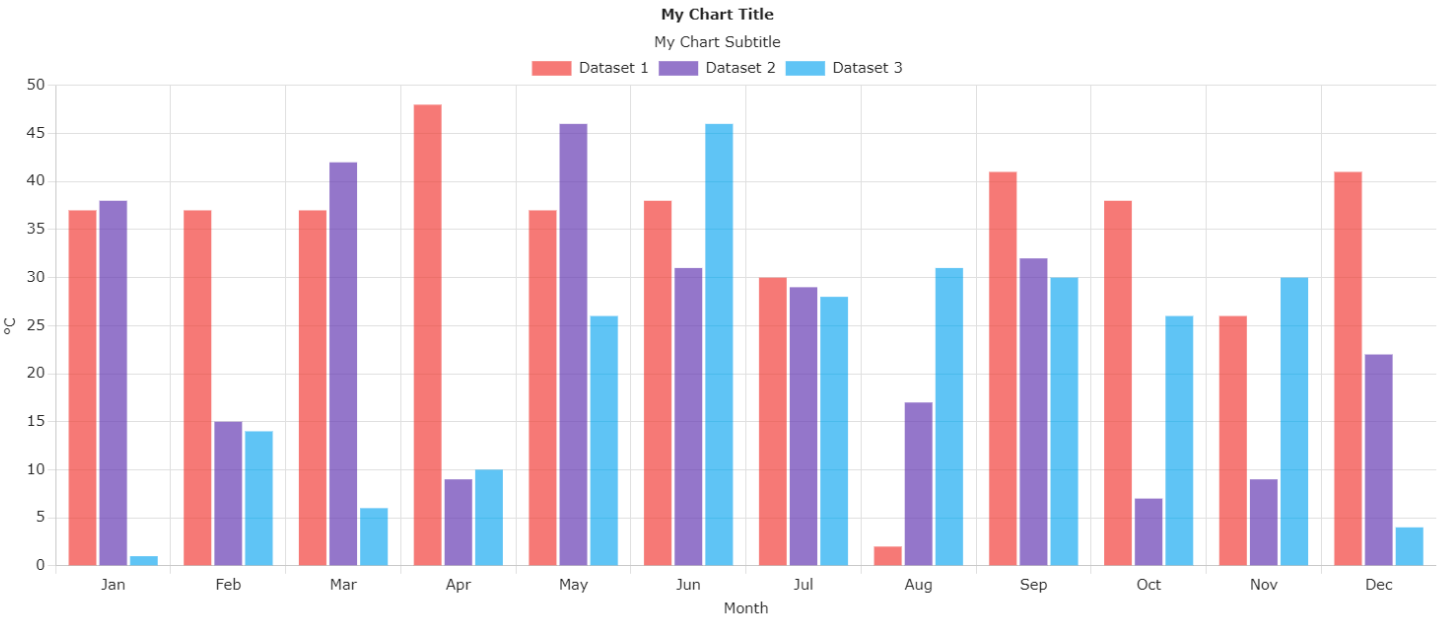


Figure 141:

Event	Description and Parameter
evClick	Triggered when the user clicks on a data element such as a bar in a bar chart. There are 2 parameters: **pDatasetIndex** - The dataset line number in the main list **pDataIndex** - The data index within the dataset. So for those supplied in rows, it will be the column number, and those supplied in lists, it will be the row number
evLegendClick	Triggered when the user clicks on a legend item. There are 3 parameters: **pDataIndex** - The data index of the data in the dataset (only for Pie, Doughnut and Polar Area) **pDatasetIndex** - The dataset line number in the main list (For all except Pie, Doughnut and Polar Area) **pHidden** - True, if the related data is now hidden

If $legendclickhidesdata is true (the default), when you click on an item in the legend it is toggled on/off and the dataset in the chart is hidden or shown; its state is reported in the pHidden parameter for evLegendClick.

Mixing Chart Types

In some cases you can mix chart types. A good use case of this is to show a line of best fit on a scatter chart. You can do this by setting the 'type' on the dataset which you wish to be different to your charts $charttype property. Here is an example of how you could achieve this:

```
Do iData.$define(lTemp,lSales)
Do iOptions.$define(Key,Value)
Do iChartList.$define(iData,iOptions)
Do iData.$add(14.2,215)

Do iData.$add(16.4,325)
Do iData.$add(11.9,185)
Do iData.$add(15.2,332)
Do iData.$add(18.5,406)
Do iData.$add(22.1,522)
Do iData.$add(19.4,412)
Do iData.$add(25.1,614)
Do iData.$add(23.4,544)
Do iData.$add(18.1,421)
Do iData.$add(22.6,445)
Do iData.$add(17.2,408)

Do iOptions.$add("backgroundColor",kJSThemeColorPrimary)
Do iChartList.$add(iData,iOptions)

Do iData.$clear()
Do iOptions.$clear()

Do iData.$add(11,150)
Do iData.$add(26,650)
Do iOptions.$add("type","line")
Do iOptions.$add("borderColor",kJSThemeColorSecondary)
Do iChartList.$add(iData,iOptions)
```

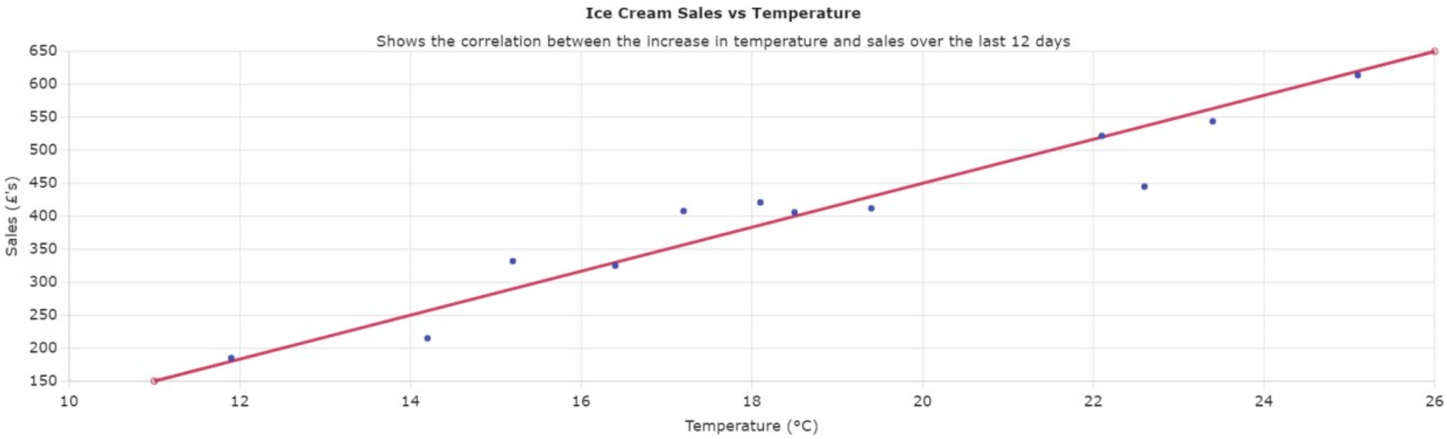


Figure 142:

Note how the second dataset, used to portray a line of best fit, is calculated manually, i.e. there is no function to calculate an actual line of best fit.

Check Box Control

Group	Icon	Name	Description
Buttons	☑ Opt1 ☑ Opt2	Check Box	Check box for on/off values

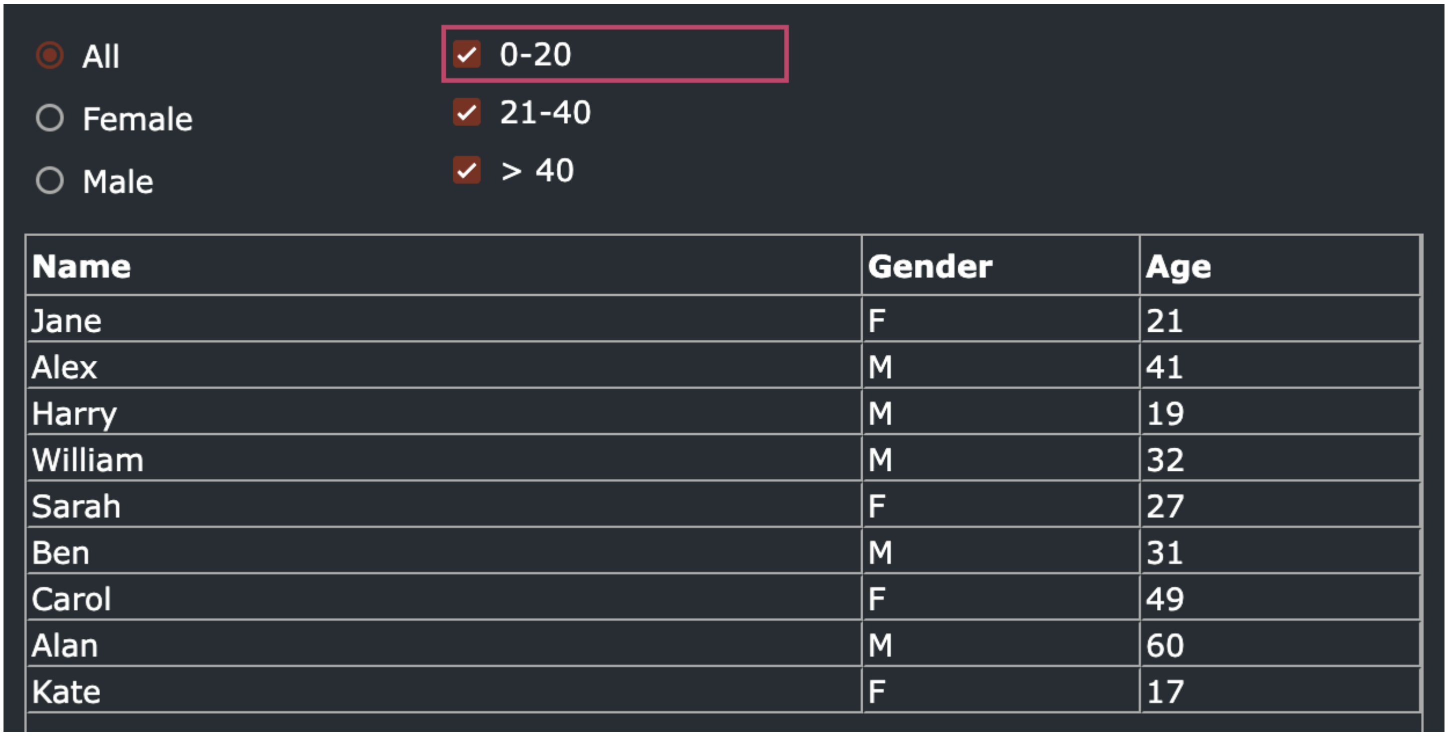
The **Check Box** control can represent On / Off or Yes / No values and is typically used to allow the end user to turn an option on or off, or accept or decline a preference. Several of the sample apps in the JavaScript Component Gallery feature check boxes, such as the Map example.

The variable you specify in the $dataname property of a Checkbox should be a Number or Boolean variable. The $text property specifies the label text for the Checkbox. The $checkboxcolor property specifies the color for the Checkbox (and check boxes when they appear in Lists, Data grids & Tree lists).

When a Checkbox is clicked an evClick event is triggered with the current value reported in the pNewVal parameter.

Example

There is an example app called **JS Radio and Checkbox** in the Samples section in the Hub in the Studio Browser to show how you can use a check box (and a Radiogroup); the same app is in the JavaScript Component Gallery. The example uses a series of check boxes and a radio button group to filter a list of people based on their gender and age group. The group of controls on the remote form could look like this (the following screen shows the 'professional' JS Theme in use):



● All
○ Female
○ Male

☑ 0-20
☑ 21-40
☑ > 40

Name	Gender	Age
Jane	F	21
Alex	M	41
Harry	M	19
William	M	32
Sarah	F	27
Ben	M	31
Carol	F	49
Alan	M	60
Kate	F	17

Figure 143:

The $dataname for each of the check boxes is iAgeRange1, iAgeRange2, and iAgeRange3 respectively. These are all Boolean variables defined in the remote form, and the $dataname of the Radio button group is iFilter, which is defined as a Short integer. Each separate check box and the Radio group has a simple event method, which is:

```
On evClick
  Do method filter
```

which will call the 'filter' class method when any of these objects is clicked. The filter method filters the contents of the list called iList based on the selection of the check boxes and radio buttons, and has the following code:

```
Do iList.$unfilter(0)

If not(iAgeRange1)
  Do iList.$filter(not(iAge<=20))
End If

If not(iAgeRange2)
  Do iList.$filter(not(iAge>=21&iAge<=40))
End If

If not(iAgeRange3)
  Do iList.$filter(not(iAge>40))
End If

Switch iFilter
  Case 1
    Do iList.$filter(iGender='F')
  Case 2
    Do iList.$filter(iGender='M')
End Switch
```

Note the 'Smart list' capability has to be enabled on the iList variable to allow the built-in filtering using the $filter method; this is done in the $construct method of the form in the example, using the following code:

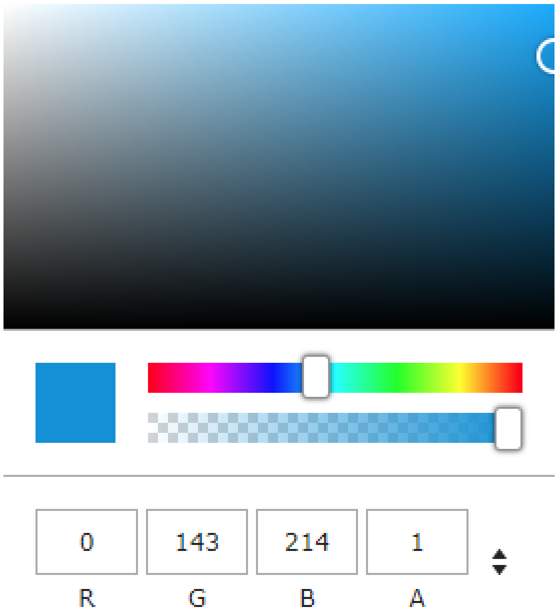
```
Do iList.$smartlist.$assign(kTrue)
```

Color Picker

Group	Icon	Name	Description
Other		Color Picker	Allows the end user to select a colo color palette, or RGB, HSL, or HEX r

The **Color Picker** component allows the end user to select a color either by sliding a color slider and clicking on the color palette, or by entering a color number in RGB, HSL, or HEX format; an alpha slider can be shown to allow the end user to select the alpha setting (transparency) for the color.

You would typically open the Color Picker in a subform or palette window, to allow the end user to select a color, then close the subform returning the selected color value to the main form to assign to an object or property. Otherwise, you could add a color picker to a general settings panel in your app, such as a side panel. The following screenshot shows the color picker with the color preview swatch, alpha slider and the format entry fields for specifying an RGBA color.



R	G	B	A
0	143	214	1

Figure 144:

The color selected in the color picker is returned to the instance variable specified in **$dataname** of the control, which must be a 64-bit integer if you want to include the alpha channel, otherwise you can use a 32-bit integer if the alpha channel is not required.

There is an example application called **JS Color Picker** in the **Samples** section of the **Hub** in the Studio Browser showing the Color Picker control, including the different color number formats and the predefined color swatches.

Properties

The Color Picker has the following properties to set up the appearance and behavior, such as showing a color swatch preview, showing the alpha slider, or controlling which color number formats are shown (RGB, HSL, or HEX).

Property	Description
$colorformats	The color formats shown in the list of color formats; if empty, the color format list is hidden so the end user cannot enter a color number. One or more of the constants: kJSColorPickerFormatRGB, kJSColorPickerFormatHex, kJSColorPickerFormatHSL (selected via a check list in the Property Manager). If multiple formats are selected, a button is shown allowing the end user to cycle through the color formats; see the example app in the Hub
$currentcolorformat	The initial color format displayed to the user; ignored if $colorformats is empty or does not include the specified format
$copybutton	If kTrue, a copy button is shown allowing the end user to copy the currently displayed color to the clipboard
$previewcolor	If kTrue, a swatch preview of the selected color is shown; if $copybutton is also kTrue, the end user can click on the color swatch to copy the color to the clipboard
$swatchlist	A list instance variable containing a single column list of colors which are added as color swatches to the bottom of the picker; if blank no swatches are added, see below
$usealpha	If kTrue (and the variable in $dataname is capable of storing a 64-bit integer), the control displays the alpha slider and value, in the range 0 (transparent) to 1 (opaque)

Events

The **evColorPicked** event is triggered when the user has selected a color, that is, when they let go of the pointer after selecting a color, or when they tab out of a color number input field. **pColor** contains a 64-bit integer representing the selected color.

The **evColorChanged** event is triggered each time the color is changed; **pColor** contains a 64-bit integer representing the selected color. If you wish to trap this event, it is recommended you use only a client-executed event handler since this will fire a lot of events as the user drags on a color slider.

The example app in the Hub uses the evColorPicked and evColorChanged events and the new *$clientevent* method. The *$event* method for the color picker control handles the **evColorPicked** event as follows:

```
On evColorPicked
  Calculate iColorPicked as pColor
```

While the *$clientevent* method for the control (which is set to execute on the client) handles the **evColorChanged** event, which changes rapidly as you click and drag inside the color palette of the control.

```
On evColorChange
  Calculate iColorChange as pColor
```

Predefined Color Swatches

You can add a number of predefined color swatches to the color picker to allow the end user to select a preset color; the color swatches could be colors defined in your corporate branding or colors that are in constant use in your app. The colors are specified in a list instance variable containing a single column list of colors which is assigned to the $swatchlist property; if empty, no swatches are added to the picker. For example, you could define the list in the $construct method of the form and assign the iswatches list to $swatchlist.

```
# Define iswatches (List), lcolor (64-bit integer)
Do iswatches.$define(lcolor)
Do iswatches.$add(rgb(0,142,214))
Do iswatches.$add(rgb(15,108,177))
Do iswatches.$add(rgb(255,155,0))
Do iswatches.$add(rgb(0,54,200))
Do iswatches.$add(rgb(225,216,29))
Do iswatches.$add(rgb(205,00,105))
```

The following screenshot shows the color picker with a set of predefined color swatches displayed at the bottom, defined in the iswatches list and assigned to $swatchlist.

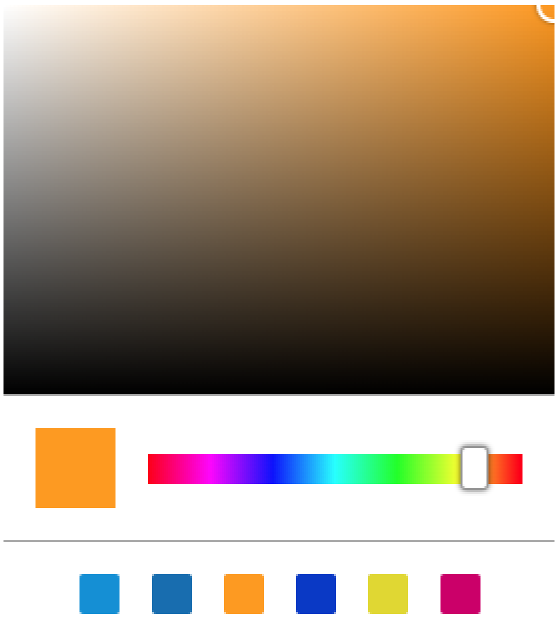


Figure 145:

Combo Box Control

Group	Icon	Name	Description
Lists		Combo Box	Field combining entry box and dro

The **Combo Box** control is a combination of a *data field* and a *dropdown list* from which the end user can make a selection or enter their own value into the field. There is an example app in the **Samples** section in the **Hub** in the Studio Browser (called **JS Droplist,**

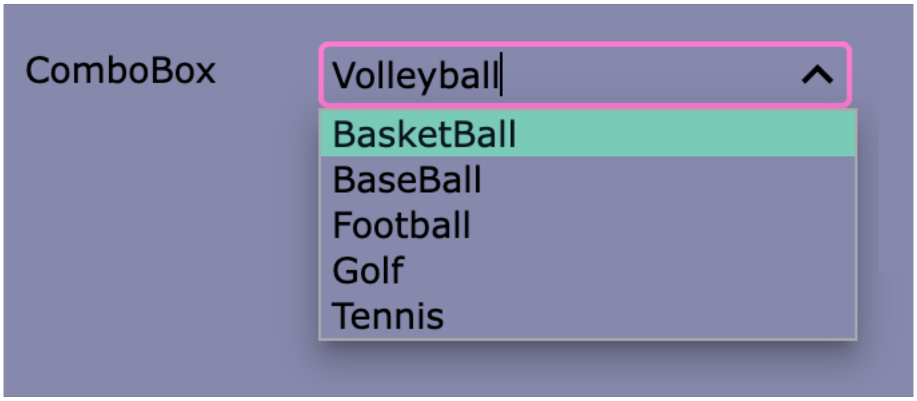


Figure 146:

Combo, Popup); the same app is available in the JavaScript Component Gallery. The following screen shows a Combo Box using the Soft JS Theme.

The variable for the data field part is specified in the $dataname property. You can specify a default list of options in the $defaulttext property, which is a comma-separated list of options, or build the list dynamically (with $::listname, see below). When $defaulttext is specified, $defaultline specifies the list line which is selected when the form is opened (set to 1 by default). The $::listheight property specifies the height of the droplist. The Combo Box has the $negallowed property which means it can display negative numbers.

Rather than using a default list specified in $defaulttext, you can assign the name of a list variable to the $::listname property to assign the contents of the list to the droplist part of the combo box; $listcolumn specifies which column of the list variable is used to populate the droplist part of the combo box.

When the list in a Combo Box is clicked an evClick is generated with the selected list line reported in the pLineNumber parameter.

Content Tips

The Combo box control has the $::contenttip property which is a text string which is displayed in the edit field part of the combo box when it is empty to help the user understand what content should be entered into the field. For example, for a Last name field you could enter 'Enter your last name' into $::contenttip to prompt the end user for their last name.

Auto correction and capitalization

Combo boxes have the **$autocorrect** and **$autocapitalize** properties, which when enabled means that any text entered into the edit field section of the control is corrected for spelling and capitalization automatically.

Example

The maintenance screen in the **Webshop** sample app allows the user to enter new products or delete existing ones: specifically, the data in the Webshop app contains food and drink items, but it could be any type of products. When the user enters a new product, they can select the product type from a Combo control; this allows the user to select from a list of given product types or enter a new one.

The $dataname of the combo control is set to iDataRow.product_group, and the $::listname is iGroupList. The evAfter event is enabled in the $events property of the control. In the $construct method of the form, the iDataRow row variable is defined from the T_Products table class, as follows:

```
# $construct of jsMaintenance form\

# sets up form sizes, etc, then…
Do iDataRow.$definefromsqlclass($tables.T_Products)
Do $cinst.$objs.$sendall($ref.$construct())
```

The last line of code triggers all the field specific $construct methods which in this case includes the Combo box control; the code defines the iGroupList from the product_group column in the T_Products table class, performs a select on the data, and fetches all the data back into the iGroupList variable.

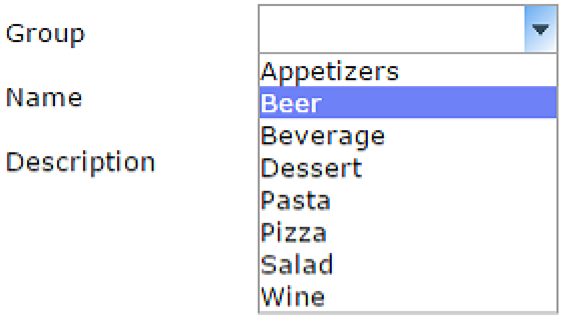


Figure 147:

```
Do iGroupList.$definefromsqlclass($tables.T_Products,'product_group')\

Do iGroupList.$selectdistinct()
Do iGroupList.$fetch(kFetchAll)
```

When the user selects an item in the list or enters a new item into the entry part of the combo box, an evAfter event is triggered and the $event method behind the combo control is called, as follows:

```
On evAfter
 Do method newItem
 Do $cinst.$setcurfield('product_name') ## puts the focus in the product_name field
```

The newItem method is placed behind the Combo box control itself and contains the following code:

```
Do iGroupList.$search($ref.product_group=iDataRow.product_group, kTrue, kFalse, kFalse,kFalse)    ## test iGro

If iGroupList.$line=0   ## if not found in the group list
  Do iGroupList.$add().product_group.$assign(iDataRow.product_group)   ## add a new group to the list
End If
```

Complex Grid

Group	Icon	Name	Description
Lists	▦	Complex Grid	Grid which can display all types of formatting

A **Complex Grid** can display multiple rows and columns of data taken from a list variable specified in the $dataname property of the control. You can use a $construct method behind the grid control itself to build the list data to populate the fields in the complex grid. To create a complex grid, you can place other controls in the row and header sections of the grid control, including standard entry fields, droplists, buttons, and check boxes: these controls are duplicated for every row in the grid, displaying each row of data from the data list. The $dataname of each component you place in the grid must correspond to a column in your list variable supplying the data to the grid. A complex grid is a *container field* having its own $objs group containing the objects inside the grid control.

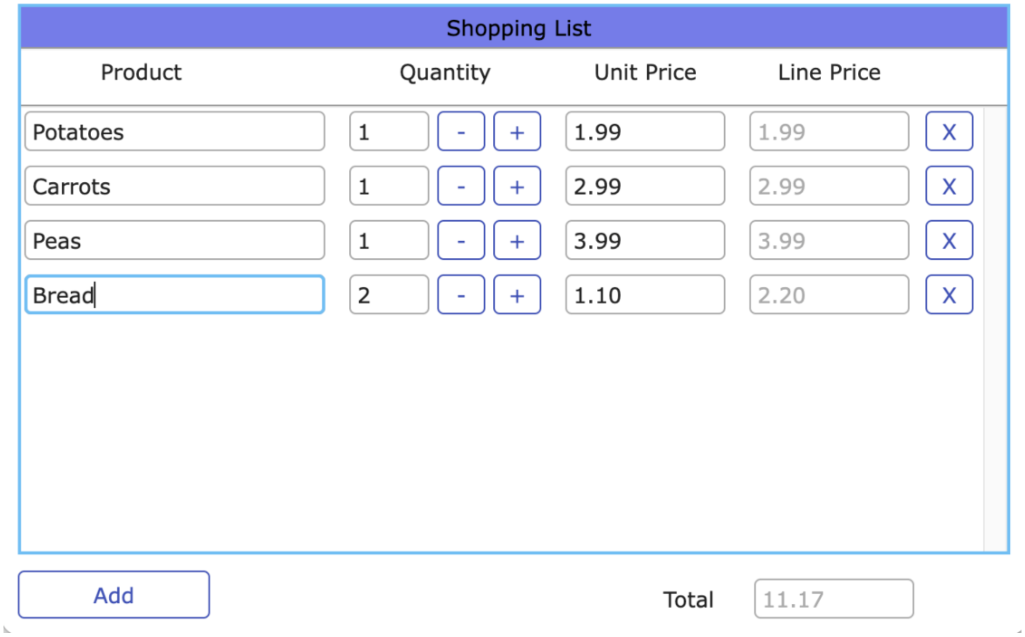
There is an example app called **JS Complex Grid** in the **Samples** section in the **Hub** in the Studio Browser; the same app is available in the JavaScript Component Gallery.

In addition, there is a **Webshop** app that uses complex grids in the **Applets** section in the **Hub** in the Studio Browser which is described later in this section.

Events

You can place event methods behind the embedded controls to react to user input and clicks within individual fields/cells in the grid. For example, you can have a button in each row of the grid which when clicked triggers an evClick event which runs the $event method for the button that performs an action based on the row clicked.

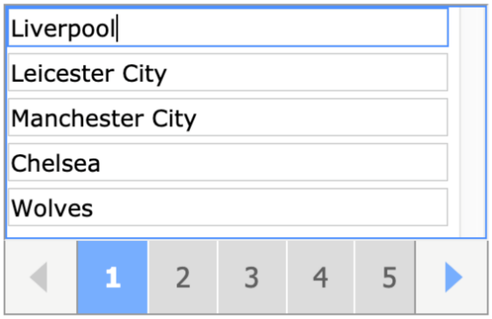
The Complex Grid itself can have evClick & evDoubleClick events. When clicking on the background of a complex grid row, or a control within the grid which does not have a click event enabled, the evClick or evDoubleClick will be fired. Both of these events



Product	Quantity	Unit Price	Line Price
Potatoes	1	1.99	1.99
Carrots	1	2.99	2.99
Peas	1	3.99	3.99
Bread	2	1.10	2.20

Total 11.17

Figure 148:



Liverpool

Leicester City

Manchester City

Chelsea

Wolves

Figure 149:

Tabbing inside a grid

When tabbing through controls on a form, the Complex grid is treated as a single tab stop (pressing Tab while the grid has focus will move the focus to the next control on the form). Pressing **Enter** while the grid has the focus will move the focus inside the grid. **Tab** and **Shift+Tab** can then be used to move between controls in the grid. Pressing **Escape** will return focus to the grid.

Scrolling

The complex grid has the properties $vscroll and $hscroll to allow you to scroll a grid dynamically. The $vscroll property takes the row number in the grid to scroll to. The $hscroll property takes the absolute horizontal pixel position to scroll to in relation to the left edge of the grid control, that is, the grid will not scroll by a specified amount, rather the grid will scroll to the absolute position in the grid specified by $hscroll.

Scroll Tips

You can use the property $vscrolltips to display a scroll tip when a complex grid scrolls vertically. If $vscrolltips is kTrue, the default scroll tip is the contents of column 1 of the list for the first fully displayed row. To override this default scroll tip, implement a client-executed method for the complex grid object, called $getscrolltip. $getscrolltip accepts a single parameter (the row number for which the scroll tip is required), and returns the scroll tip text.

Scrollable footer

A complex grid can include a scrollable footer section similar to the existing scrolling header section. To enable a scrollable horizontal footer, you need to set $showhorzfooter to true. The complex grid has the following properties to control the appearance of the footer:

- **$horzfooterheight**
  The height of the grid horizontal footer
- **$horzfooterfillcolor**
  The fill color for the grid horizontal footer
- **$horzfooterborder**
  The border style for the grid horizontal footer
- **$horzfooterlinestyle**
  The line style for the grid horizontal footer

Row divider line style and Field Styles

The $rowdividerlinestyle is assignable at runtime and by $fieldstyle. As $rowdividerlinestyle is a custom field in a $fieldstyle it gets assigned at runtime, and is treated like any other runtime property change, therefore it is assignable at runtime. Note that $rowdividerlinestyle changes just the border between each row in a Complex grid, unless $rowborder is set to kJSborderPlain, in which case it also effects the border around the client, i.e. the section of the complex grid which contains the rows.

Extra space

The $extraspace property can be used to add extra space to the line content in the grid. If $extraspace is zero, the height of each row is the default height of the row content. If $extraspace is greater than zero, the height of each row is the font height + $extraspace.

Dropping data onto a grid

The end user can drag data from a remote form field and drop it onto a cell in the grid. Complex grids have the evCanDrop and evDrop events, and the $dropmode property to enable drop support. The pDropRow event parameter is available for evCanDrop, evWillDrop and evDrop events, and reports the row of the complex grid on which the drop is to occur (zero if the control does not belong to a complex grid).

It is possible to drop data onto a single control in the grid (a cell) in any row in the grid, as long as it has its $dropmode enabled. If not, the complex grid itself will receive the drop.

Exceptions

You can format individual cells in a complex grid by applying “exceptions” to those cells: you can then apply different formatting to those cells. For example:

```
Calculate $cinst.$objs.Products.$objs.Product.6.$backcolor as kBlue
```

or using indirection:

```
Calculate lNum as 4

Calculate lProp as "$backcolor"
Calculate $cinst.$objs.Products.$objs.Product.[lNum].[lProp] as kBlue
```

You can attempt to set an exception for any property, although in practice this may not be satisfactory for some properties. Appearance properties, and button text for example should however all work as expected.

You can set exceptions in both server and client executed code.

In addition, the method $clearexceptions() can be used to clear exceptions. For example (Products is a complex grid object):

```
# Clear all exceptions for the Product object on all lines where it has exceptions\
Do $cinst.$objs.Products.$objs.Product.$clearexceptions()

# Clear all exceptions for the Product object on line 4
$cinst.$objs.Products.$objs.Product.$clearexceptions(4)

# Clear all exceptions set in the complex grid
Do $cinst.$objs.Products.$clearexceptions()

# Clear all exceptions set on line 4 of the complex grid
Do $cinst.$objs.Products.$clearexceptions(4)
```

You can execute $clearexceptions() in both server and client executed code.

Existing users should note: Prior to the implementation of exceptions, objects in the row section could lose property values set at runtime, when updating the grid data. This issue has been resolved as part of the exception implementation.

Complex Grid Restrictions

A complex grid cannot contain another complex grid as a member, in any section. A complex grid cannot contain a subform in the row section. These restrictions apply to controls that would be direct members of the section, or indirect members that are children of a paged pane. Omnis enforces this by preventing you from dropping these controls into the relevant section(s).

Note: the $add() method for remote form class objects has not been updated to enforce this restriction, therefore using $add() to place controls in a section which does not support them could lead to undesirable results.

Example

The **Webshop** sample app, available in the Hub in the Studio Browser, uses a **Complex grid** in the main product remote form to display a list of products. Individual fields for the picture, name, description, price/size of the product are added to the first line of the complex grid; when the form is opened on the client and the data is loaded into the grid, these fields and are repeated *for each row* in the data list (one row per product).

The $dataname of the complex grid is set to iProductList which is built from a table class T_Products which is linked to a schema class sProducts. A $construct method is placed behind the complex grid that builds the list needed for the complex grid data.

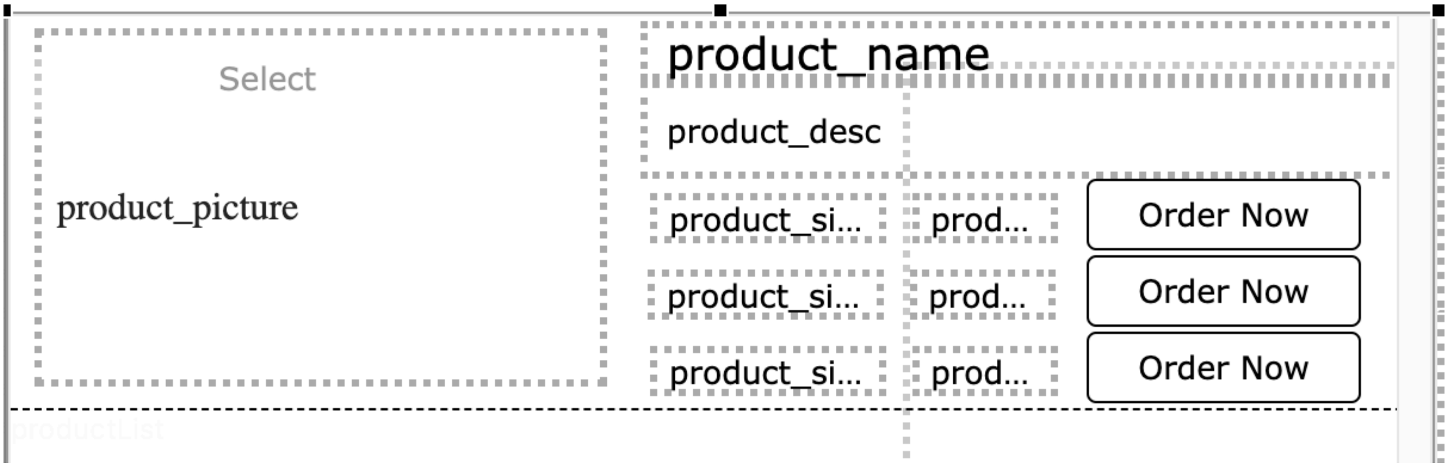


Figure 150:

```
# $construct of complex grid control in jsShop form

Do iProductList.$definefromsqlclass($tables.T_Products)
Calculate whereClause as con('WHERE product_group = ',kSq,'Appetizers',kSq)
Do $cfield.$build(whereClause)         ## calls $build

# $build method also behind complex grid control in jsShop form
Do iProductList.$select(pWhereClause,' ORDER BY product_isfood desc')
Do iProductList.$fetch(kFetchAll)
```

When the form is opened, the $construct method is run and the product list is built from the database, while the data itself is displayed in the various fields embedded in the complex grid with each product shown on a separate line in the complex grid.

There are three order buttons placed in the row of the complex grid; they are repeated for each product in the list and allow the end user to order different sizes of product, such as a small, medium, or large drink or pizza. Each of the buttons has a simple method behind it that passes a number to the process_order class method; the first button sends value 1, the second button value 2, and the third button value 3.

```
# 'Order now' button method
On evClick
  Do method process_order (1)
```

See the *Data Grid* section for the process_order method which updates the iOrderList and the Orders data grid accordingly.

Data Grid Control

Group	Icon	Name	Description
Lists	▦	Data Grid	Simple grid for text and numerical display

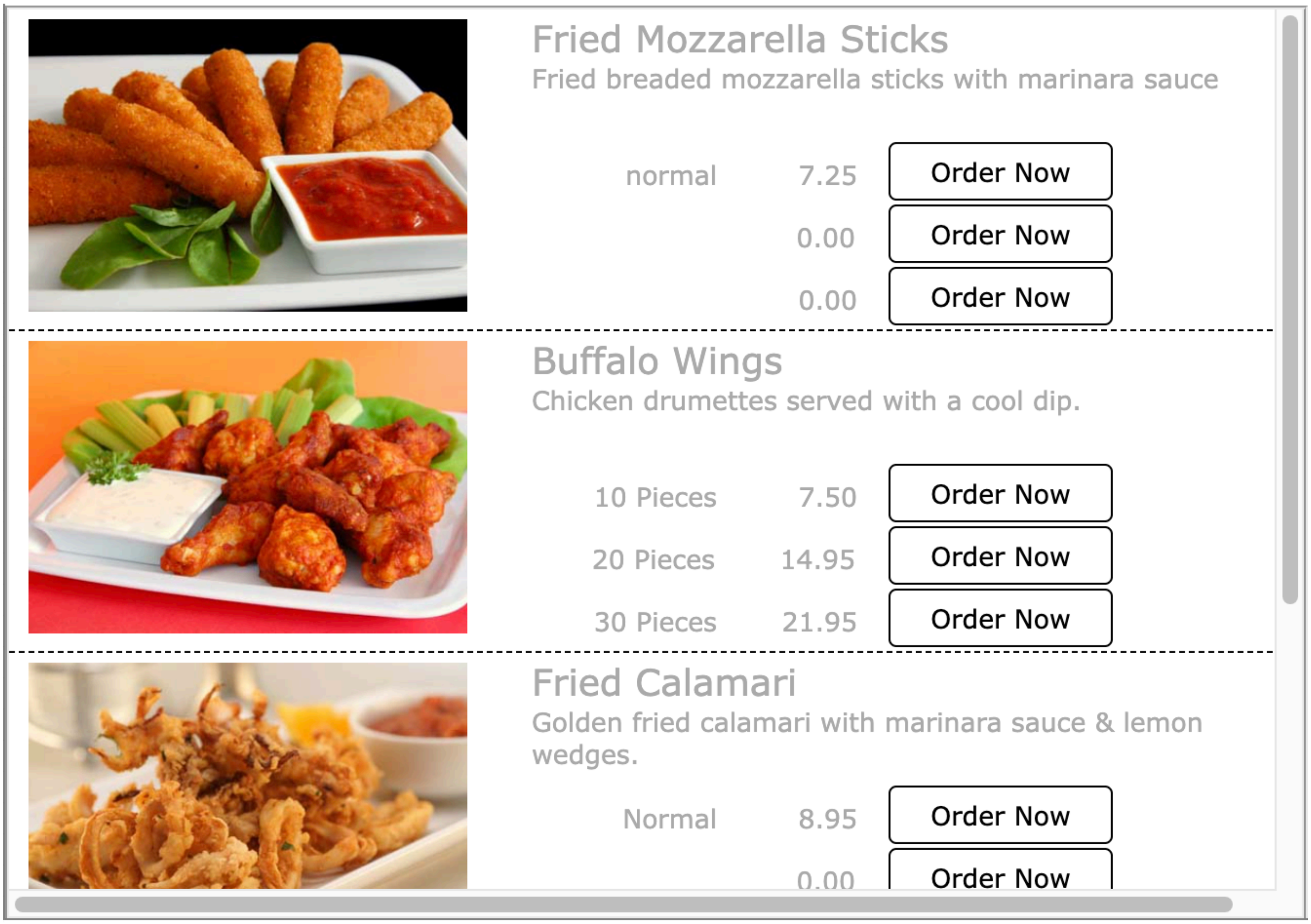
The **Data Grid** is a powerful and versatile control that can display character and numeric data in a grid like structure, much like a table or spreadsheet format, allowing you to create compact, data-rich UIs for your web and mobile applications.

There is an example app called **JS Data Grid** in the **Samples** section in the **Hub** in the Studio Browser (the same app is available in the JavaScript Component Gallery), showing how to use Data grids, including how to use custom cell formatting; the following screen shows the Vintage JS Theme in use.

In addition, the **Webshop** example under the **Applets** section in the **Studio Browser** uses a data grid which is described later in this section.

Setting up a Data Grid

The content for a data grid is supplied from a *list variable* specified in the **$dataname** property. The number of columns in your grid



Fried Mozzarella Sticks

Fried breaded mozzarella sticks with marinara sauce

normal 7.25

0.00

0.00

Buffalo Wings

Chicken drumettes served with a cool dip.

10 Pieces 7.50

20 Pieces 14.95

30 Pieces 21.95

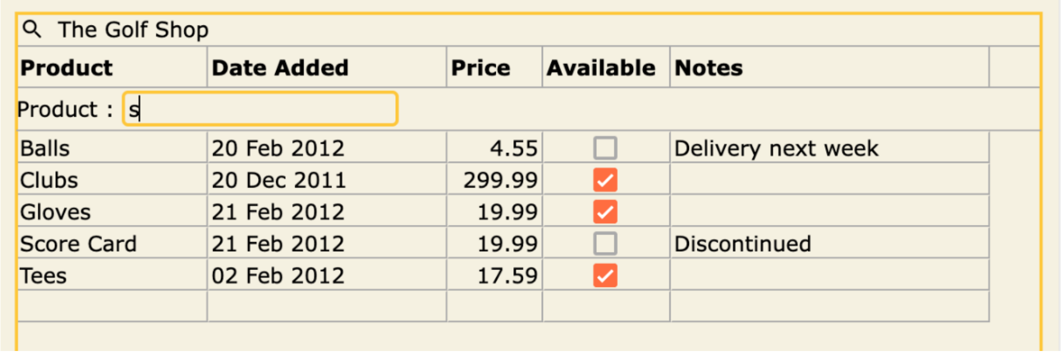
Fried Calamari

Golden fried calamari with marinara sauce & lemon wedges.

Normal 8.95

0.00

Figure 151:



The Golf Shop

Product : s

Product	Date Added	Price	Available	Notes
Balls	20 Feb 2012	4.55	☐	Delivery next week
Clubs	20 Dec 2011	299.99	☑	
Gloves	21 Feb 2012	19.99	☑	
Score Card	21 Feb 2012	19.99	☐	Discontinued
Tees	02 Feb 2012	17.59	☑	

Figure 152:

The end user can enter data into the cells of the grid if $enterable is enabled, and as the end user tabs the grid can grow by adding more lines to accommodate more data if $extendable is enabled. If $autoedit is true, and a cell is editable, it will automatically go into edit mode when it is selected (and $hcell or $vcell are set).

The height of the rows in a data grid adjusts to fit the height of the font size specified in $fontsize, unless you specify a fixed height in pixels in $::rowheight. In addition, the height of the header area is adjusted automatically according to the font, but you can fix the height in $::headerheight.

Events

Along with the standard evClick and evDoubleClick events the Data Grid reports a number of events which you can detect in an event handling method behind the grid control.

- **evClick** and **evDoubleClick**
  sent after the data grid has been clicked or double-clicked
  pHorzCell - The column number of the new current cell.
  pVertCell - The row number of the new current cell.
  pDataColumnName - the data list column name (or number) when the event is triggered

- **evCellChanged**
  sent when the current cell has changed, e.g. when navigating between cells with the arrow keys or clicking a cell that isn't the current cell.
  pHorzCell - The column number of the new current cell.
  pVertCell - The row number of the new current cell.
  pDataColumnName - the data list column name (or number) when the event is triggered

- **evCellValueChanged**
  sent when the user has changed the value of a cell.
  pHorzCell - The column number of the cell in the grid that has changed (not the column of the data list belonging to the data grid)
  pVertCell - The row number of the cell in the grid that has changed.
  pDataColumnName - the data list column name (or number) when the event is triggered

- **evCellValueChanged**
  sent when the user has changed the value of a cell.
  pHorzCell - The column number of the cell in the grid that has changed (not the column of the data list belonging to the data grid)
  pVertCell - The row number of the cell in the grid that has changed.
  pDataColumnName - the data list column name (or number) when the event is triggered pIsNewRow - if true, the cell belongs to a new row

- **evColumnsResized**
  sent when the user has resized a column.
  pColumnWidths - a comma separated list of the new column widths

evCellChanged

When evCellChanged is triggered, pVertCell will be the next line number in the list, but at the point when the event is triggered, pVertCell will reference a line which does not exist yet, which may cause an issue in the code in your event method. To mitigate this, you should check pVertCell is valid before executing the other list code.

pDataColumnName

The pDataColumnName event parameter contains the data list column name (or number) when the event is triggered. This is useful when columns in the data list do not map directly to the columns of the form data grid, that is, if $columndatacol is used to set the column order. If the list column does not have a name, the parameter contains 'C1', 'C2', etc, so it can be used notationally. The value of the cell can be obtained with: iDataList.[pVertCell].[pDataColumnName].

Sorting Grid Columns

The data in a data grid is not sorted by default – initially the order of the data in a data grid is the same as set by the data list specified in $dataname, and depends on how and in what order the data is compiled. However, you can use the $sort() method to sort the data on a specified column in the data list, for example, you can include $sort in your method that builds the data list. The syntax of the $sort() method is:

```
Do list.$sort(calculation | sort field[,bDescending=kFalse]...)
```

The calculation or sort field can use a colname, $ref.colname or listname.colname to reference the column in the data list to be sorted. The data is sorted in ascending order by default, but you can pass the bDescending parameter as kTrue to sort the data in descending order. You can specify up to nine sort fields, including the sort order flag.

The end user can sort the data in a grid by clicking on a column header, when $cansortcolumns is set to kTrue (the default for new data grids). The grid data is sorted in ascending order, and an arrow icon is displayed in the column header to indicate the sort order; clicking again *on the same column header* will reverse the sort order on that column, so after a second click the sort order will be descending. Clicking on a different column header will sort the grid on that column, and the icon will move to that column.

The $cansortcolumns property enables the ability to sort all columns in the grid, but you can use the $columncansort property (under the Column tab in the Property Manager) to enable or disable sorting for each column (when $cansortcolumns is kTrue); note you have to set $currentcolumn to apply properties to specific columns.

Customizing the sort order

If you wish to customize or override the default sort order when the end user clicks the header of a sortable column, you can use the client method **$sortgrid**(pColumnNumber,pDescending), which can be added to the control methods for the data grid and must be client executed. The method has two parameters:

- **pColumnNumber**
  the number of the column in the $dataname list which should be sorted

- **pDescending**
  True if the sort should be in descending order, false for ascending

This method will be called whenever the user clicks on a column header to sort the list, and so can contain any code to implement your own sort. The method should return kTrue to indicate that you have performed a custom sort. Returning kFalse (or nothing), will trigger the default sort order to be used.

Enter Key Behavior

When **$entertodoubleclick** is true, the Enter key is *interpreted as a double-click* on the data grid. In this case, a double-click event is sent when the focus is on the grid and the Enter key is pressed, allowing more control from the keyboard for the lists and grid controls. The property is available for Data grids, as well as Lists, Tree lists, and the Date picker control.

The property is set to kFalse by default (to maintain backwards compatibility), other than for JS Lists, which defaults to kTrue which interpreted Enter as a double-click in previous versions.

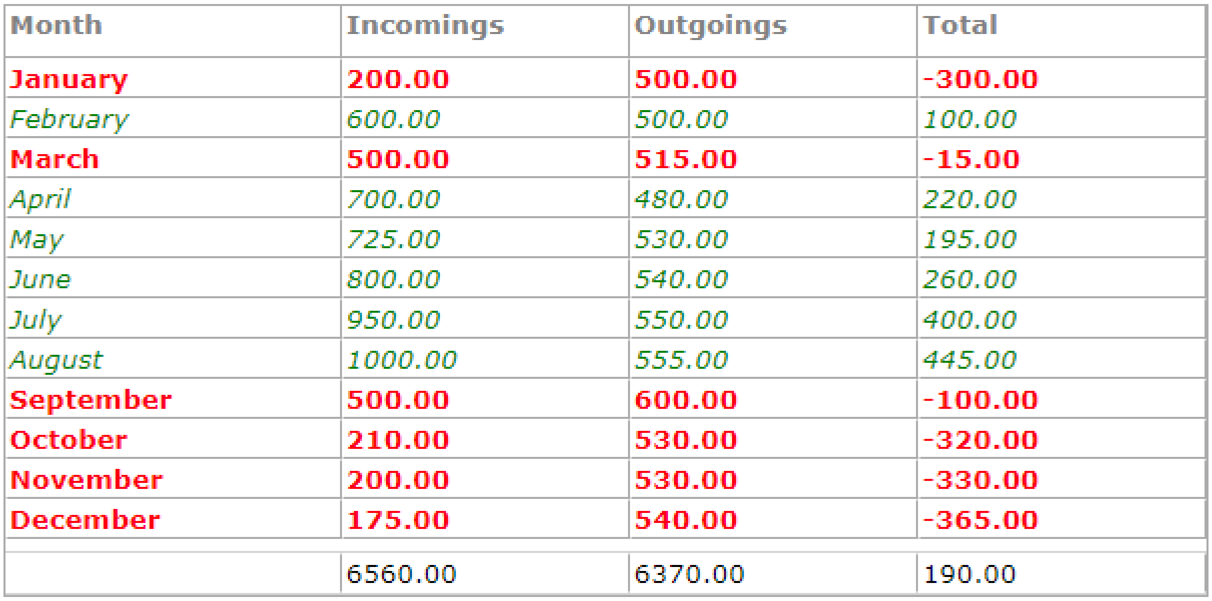
Column Width

The **$::columnwidths** property allows you to set up the widths of the columns, which is a comma separated list of integer values representing each column width in pixels. In order for the data grid to cater for multiple screen sizes, the $columnwidthsarepercentage property allows you to switch to using a percentage in the $::columnwidths property. If true, the column widths in the data grid specify a percentage of the width of the control rather than a specific number of pixels. This affects the properties $columnwidth, $::columnwidths, and $columnminwidth.

The **$resizecolumn** property allows you to specify the column number of the column that is resized when the width of a data grid changes: zero means the last column extends if necessary to fill the control width, -1 means no column is resized. The property does not apply if $columnwidthsarepercentage is kTrue.

Footer row

You can add a *Footer Row* to a Data Grid which would allow you to display column totals, for example, or any other data. The **$hasfooter** property enables the footer row for the data grid, which is a fixed, non-scrolling row at the bottom of the grid. For numeric columns, the total for each column is shown in the footer row automatically.



Month	Incomings	Outgoings	Total
January	200.00	500.00	-300.00
February	600.00	500.00	100.00
March	500.00	515.00	-15.00
April	700.00	480.00	220.00
May	725.00	530.00	195.00
June	800.00	540.00	260.00
July	950.00	550.00	400.00
August	1000.00	555.00	445.00
September	500.00	600.00	-100.00
October	210.00	530.00	-320.00
November	200.00	530.00	-330.00
December	175.00	540.00	-365.00
	6560.00	6370.00	190.00

Figure 153:

Footer Row Properties

The $hasfooter property enables the footer row for a Data grid. You can set the properties of the footer row on the Footer tab in the Property Manager using the following properties:

Property	Descriptio
$footerbackalpha	The footer background alpha
$footerbackcolor	The footer background color
$footerdateformat	The footer date format
$footerdateformatcustom	The footer custom date format
$footerfontstyle	The footer text style
$footerheight	The footer height
$footerjst	The footer text justification
$footerlabel	The default label shown in each footer cell; there is an option to pass in a placeholder
$footernumberformat	The footer number format
$footertextcolor	The footer text color
$footertype	Footer type, a kJSDataGridFooterType... constant: kJSDataGridFooterTypeTotal (the default), kJSDataGridFooterTypeMean (average), kJSDataGridFooterTypeMedian, kJSDataGridFooterTypeFooterColumnValue, kJSDataGridFooterTypeCustom, kJSDataGridFooterTypeNone
$footerzeroshownempty	If true, show as empty for zero values

Footer Column Properties & Events

The following are properties for a column in the footer row, shown under the Column tab in the Property Manager. You can set $currentcolumn (a design property) or click on a column in design mode to set the properties for each footer column.

Property	Description
$footercolumnbackalpha	The footer column background alpha
$footercolumnbackcolor	The footer column background color
$footercolumndateformat	The footer column date format
$footercolumndateformatcustom	The footer column custom data format
$footercolumnfontstyle	The footer column font style
$footercolumnhidden	If true, the column is hidden
$footercolumnjst	The footer column text justification
$footercolumnlabel	The default label shown in the footer cell
$footercolumnnumberformat	The footer column number format
$footercolumntextcolor	The footer column text color
$footercolumntype	The default is to use setting in $footertype, otherwise an individual column can be set to a kJSDataGridFooterType... constant; see $footertype above
$footercolumnvalue	If the $footercolumntype is set to kJSDataGridFooterTypeFooterColumnValue, it will use the value in $footercolumnvalue property, with the $footercolumnlabel prefix, if used. Otherwise, this property could be read (on the client) for any footer type
$footercolumnzeroshownempty	If true, the footer column is shown as empty for zero value

If a List Pager is used or a filter is applied to the list, the automatic totals are for only the data that is currently displayed; specifically for a list pager, the column totals are for the current page only.

Footer Row Methods

When $footertype is set to kJSDataGridFooterTypeCustom, you can use the **$updatefooterrow()** client method to update the contents of the footer row. The method is called to notify of the footer data changing/requiring a change for every footer column type except kJSDataGridFooterTypeNone (as there is no footer data to be displayed). It is also called on initialization, and whenever any of the data in the grid changes.

Where applicable, a column's $footercolumnvalue property will be updated before calling into $updatefooterrow() so that this value can be read and used elsewhere, i.e. when the footer type is kJSDataGridFooterType(Total/Meann).

The **pColumn** parameter is the column number in the data grid list variable (assigned to $dataname), ignoring the display order; the **pDataColumnName** is the column name in the list data.

If Character data is returned, it will be displayed as-is, and will ignore $footerlabel. If Number data or a Date is returned, it will be displayed with the $footerlabel and formatted according to $footercolumnnumberformat or $footercolumndateformat.

Events

The **evFooterClick** event is triggered if the end user clicks a cell in the footer row, passing the column number clicked.

The **evFooterUpdated** event is triggered when a footer or multiple footers have changed. pUpdatedColumns is a list of column numbers that have been updated. The event is triggered *after footers have updated* so you can receive the value of $footercolumnvalue if required.

Cell Formatting

You can apply your own formatting to individual cells in a Data Grid. For user-defined Data Grids (where $userdefined is set to true) and where a column has its columnmode set to kJSDataGridModeCustomFormat, you can customize the HTML used to layout or format an individual cell in the grid.

When the grid is rendered (this occurs on demand, e.g. when scrolling to make new data visible), it calls the object client method:

```
$formatcell(pListLineNumber, pDesignGridColumnNumber, pDataColumnNumber, pDataColumnName)
```

which you implement to return html for the formatted cell contents. The parameters pDataColumnNumber and pDataColumnName allow you to locate the correct data in the underlying list by directly referencing the column in that list, instead of the displayed datagrid.

The HTML can, within reason, be anything you like: you can also just return a text string. To assist this, there are two new calls you can make:

- styledtohtml(text)
  the styledtohtml(*text*) function returns the HTML representing the *text* string containing embedded styles inserted using *style()* function. This is a built-in function, under the General tab of the Catalog, which must be run in a client-executed method in the JavaScript Client only.
- $addcolorcss(cClassName,iRgbColor,iAlpha)
  is a method of the data grid object (as such it can only be executed in client-executed methods). Call this in $init to add your own background color class to use with the html returned by kJSDataGridModeCustomFormat.This class takes browser specific issues with transparency into account.

For example, you can use the following method:

```
Do $cinst.$objs.datagrid.$addcolorcss("myclass",rgb(255,0,0),128)
```

in $init, and then do the following in $formatcell:

```
Calculate columnvalue as iList.[row].[col]
If (columnvalue = "bad")
  Quit method con('<div class="myclass" style="padding:0;margin:0;height:16px">',columnvalue,' ','</div>')
End If

Quit method columnvalue
```

Using transparency in the CSS background allows the selection color to show through the formatted cell.

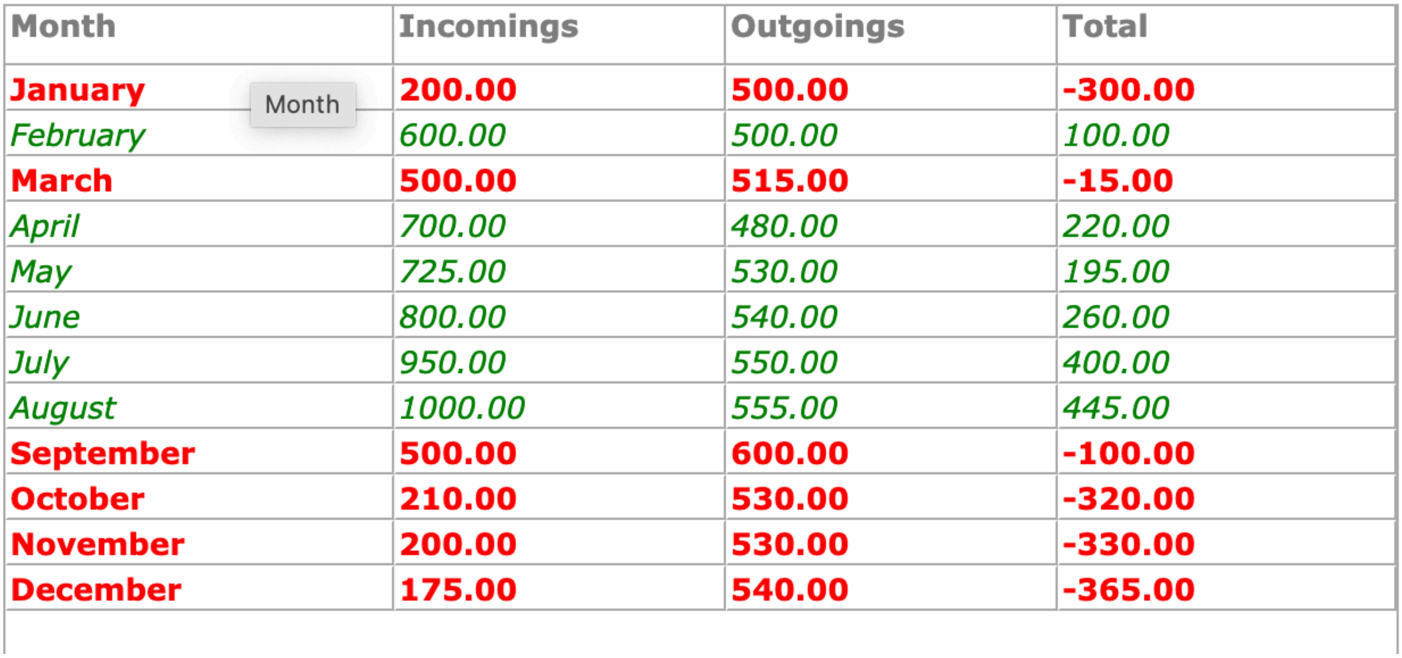
There is an example app called **Data Grid Formatting** in the **Samples** section in the **Hub** in the Studio Browser to show how you can use the grid formatting; the same app is available in the JavaScript Component Gallery.

Data Grid Filter

You can add a filter to a data grid by enabling the $hasfilerarea property. When true, the grid has a filter area which can be opened by clicking on a 'spyglass' button in the data grid header; the search filter will be applied to the current column (the Product column in the screen shown below). The end user can type into the filter entry box to filter the contents displayed in the column. The following image shows the filter enabled for column 1:

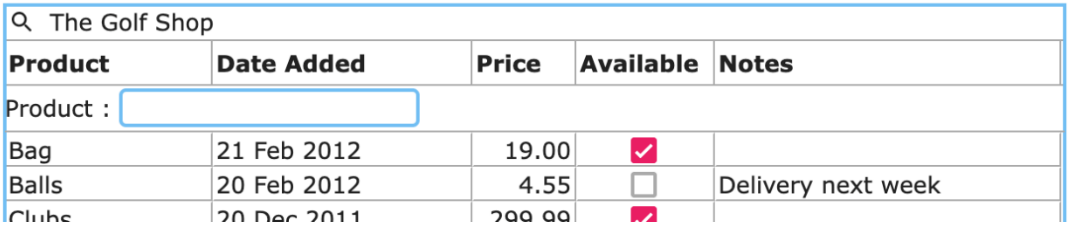
The other properties to set up the filter include:

- **$filtercol**
  The grid column to which the filter will apply; this can be changed at runtime in a client method, see below
- **$filterareaheight**
  The height of the filter area (when $hasfilerarea is enabled); if zero, the height is calculated automatically



Month	Incomings	Outgoings	Total
January	200.00	500.00	-300.00
February	600.00	500.00	100.00
March	500.00	515.00	-15.00
April	700.00	480.00	220.00
May	725.00	530.00	195.00
June	800.00	540.00	260.00
July	950.00	550.00	400.00
August	1000.00	555.00	445.00
September	500.00	600.00	-100.00
October	210.00	530.00	-320.00
November	200.00	530.00	-330.00
December	175.00	540.00	-365.00

Figure 154:



The Golf Shop

Product	Date Added	Price	Available	Notes
Product :				
Bag	21 Feb 2012	19.00	☑	
Balls	20 Feb 2012	4.55	☐	Delivery next week
Clubs	20 Dec 2011	299.99	☑	

Figure 155:

- **$filterlabel**
  The text label for the filter entry field

- **$filtervalue**
  The name of an instance variable that contains the value used for filtering

- **$multifilters**
  enables a filter for all the columns in the data grid; see below

The following code is the $event method for a data grid, set to execute on the client:

```
On evCellChanged
  Calculate iHorzCell as pHorzCell
  Calculate iVertCell as pVertCell

On evHeaderClick
  Calculate $cobj.$filtercol as pHorzCell
  # assigns the filter to the selected column
  Calculate lColNames as $cobj.$columnnames
  Calculate lColIndex as pHorzCell-1

  # Find the display name of the column clicked:
  JavaScript:lFilterName = lColNames.split(",")[lColIndex];
  Calculate $cobj.$filterlabel as con(lFilterName,": ")
```

Filter control method

You can implement the $filtergrid control method on a data grid, which will be called whenever the user types text into the filter box for a single filter enabled grid (if enabled); this control method does not apply when $multifilters is enabled.

The $filtergrid data grid method should be set to execute on the client.

```
$filtergrid(Column data, Filter String)
```

The $filtergrid method receives two parameters: ColumnData & FilterString, the data in the filtered column for the row in question and the current filter string.

You can return true to say that the row should be included, or false to exclude it. If you return null (or nothing), the default handling will be applied to determine if the row should be shown.

Open filter method

The $openfilter client-executed method can be called from $init to allow you to open the filter area in the grid when the form is opened.

- **$openfilter**([bOpen])
  opens or closes the filter area if the grid has one, and returns kTrue if the operation was completed. bOpen: Use kTrue to open the filter area or kFalse to close it. Defaults to kTrue if unspecified.

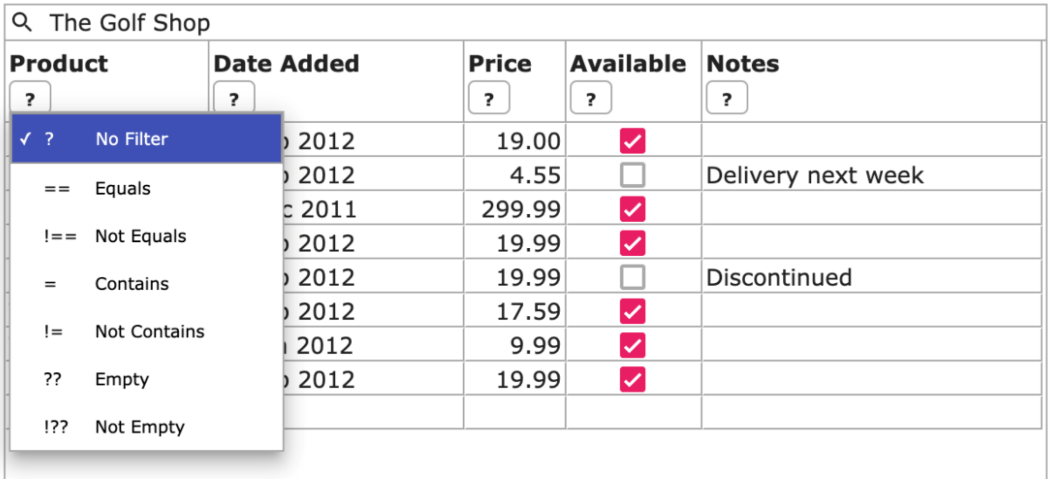
Multiple filters

In addition to being able to set a single filter for any column, you can enable a filter for all the columns in the grid by enabling the **$multifilters** property (and in this case $filtercol is ignored). When $multifilters is enabled, a default filter is added to each column in the grid (when the spyglass is clicked).

The filter has a number of operators to allow the end user to select the search type; the list of operators in the filter menu is determined by the data type of the column. When a search type is selected, a search field is displayed allowing the end user to add a search string; for date columns a date picker is also displayed.

The end user can use various keys to navigate the filter menus.

- **Tab** or **Shift Tab** will jump to the next or previous column filters.



Date Added	Price	Available	Notes
2012	19.00	☑	
2012	4.55	☐	Delivery next week
2011	299.99	☑	
2012	19.99	☑	
2012	19.99	☐	Discontinued
2012	17.59	☑	
2012	9.99	☑	
2012	19.99	☑	

Figure 156:

- **Space** or **Return** on a filter type button will reveal the context menu.
- **Up** or **Down** arrow to move up and down the filter menu items.
- **Escape** closes the filter menu, returning to filter button.
- **Space** or **Return** on a filter menu item will select the item.

By default a column will show a full set of filter types based on the column data type. You can override this and only show a subset, or remove the filter from the column altogether. You can do this using a client method called $multifiltermenu(pCol), which is called before a popup filter menu is shown for a column. You can assign the filter types for each column using the pCol parameter; the filter types can be summed for multiple filter types, for example:

```
If pCol=6
  Quit method kJSDataGridFilterContains + kJSDataGridFilterNotContains
Else If pCol=7
  Quit method kJSDataGridFilterHidden
End If
Quit method kJSDataGridFilterDefault
```

If you assign any filter types that are not supported by the column data type, the filter types will be ignored. You can use the following filter type constants:

Constant	Description
kJSDataGridFilterContains	Include contains (case insensitive)
kJSDataGridFilterDefault	Include all filter types for the column data type
kJSDataGridFilterEmpty	Include must be empty
kJSDataGridFilterEquals	Include equals (case insensitive)
kJSDataGridFilterHidden	Hide the column filter at runtime
kJSDataGridFilterLessThan	Include less than
kJSDataGridFilterLessThanEqual	Include less than or equal
kJSDataGridFilterMoreThan	Include more than
kJSDataGridFilterMoreThanEqual	Include more than or equal
kJSDataGridFilterNotContains	Include does not contain (case insensitive)
kJSDataGridFilterNotEmpty	Include must not be empty
kJSDataGridFilterNotEquals	Include does not equal (case insensitive)

Localization

The filter menu name and filter text signs can be localized (e.g. == for equals) and are stored in the Omnis omnisloc.js. The menu name and sign are separated with a “:” colon character and both parts are required.

```
"ctl_dgrd_filter_none": "No Filter:?",
"ctl_dgrd_filter_equals": "Equals:==",
"ctl_dgrd_filter_notequals": "Not Equals:!==",
```

Dynamic Filters

You can set or read the filters in a Data grid dynamically in your code using the $::filters property. The evFilterChanged event reports when data grid filters change.

The **$::filters** property (runtime only) allows you to manage the current filters for a data grid. When reading the property, the property returns a list in the same format as pFilters in evFilterUpdated (see below). Setting the property should be done using a list in the same format, that is, a single column list containing rows of data to change the filter values. The only difference for setting the filter is that only three columns are required in each row, either colNumber or colName can be supplied. If both are supplied, colName will take precedence over colNumber in the case that they reference different columns. The columns in the row can be in any order, but their names must match the format (i.e. colNumber/colName, filterType and value).

The **evFilterUpdated** event is fired every time the filter in the data grid changes. The event is fired for both types of filter ($multifilter can be kTrue or kFalse). It receives one parameter, pFilters, which contains a single column list of rows. Each row contains four columns:

- **colNumber** or **colName**
  The list column number or name.
- **filterType**
  The filter type, a kJSDataGridFilter... constant.
- **value**
  The value for the filter. The data type of this should be determined by the column data type.

When $multifilter is kFalse, pFilters is in the same format, but will only contain one row entry.

The following example filters the data grid by product and price:

```
Do lFiltersList.$define(lFilterRow)
# Filter the column iProduct by values containing "ba"
Do lFilterRow.$define()
Do lFilterRow.$addcols("colName", kCharacter, kSimplechar, 100, "filterType", kInteger, k32bitint,, "value", k
Do lFilterRow.$assigncols("iProduct",kJSDataGridFilterContains,"ba")

Do lFiltersList.$add(lFilterRow)

# Filter column 3 by values greater than 10
Do lFilterRow.$define()
Do lFilterRow.$addcols("colNumber", kInteger, k32bitint,, "filterType", kInteger, k32bitint,, "value", kIntege
Do lFilterRow.$assigncols(3,kJSDataGridFilterMoreThan,10)
Do lFiltersList.$add(lFilterRow)
Do $cinst.$objs.DataGrid.$::filters.$assign(lFiltersList)
```

Row Formatting

The $setlineheight property allows you to center text vertically in the rows in the data grid. If true, the grid sets the line height so that text is vertically centered in each row (the default is kFalse).

Header Formatting

You can create your own formatting for column headers by adding a client-side method called $formatheader which takes two parameters:

- **Parameter 1**
  the text for the column header

- **Parameter 2**
  the design grid column number (1-n)

The return value is HTML to use for the header, for example, for a bold header:

```
<b>Param 1</b>
```

For red text:

```
<span style="color:red">Param 1</span>
```

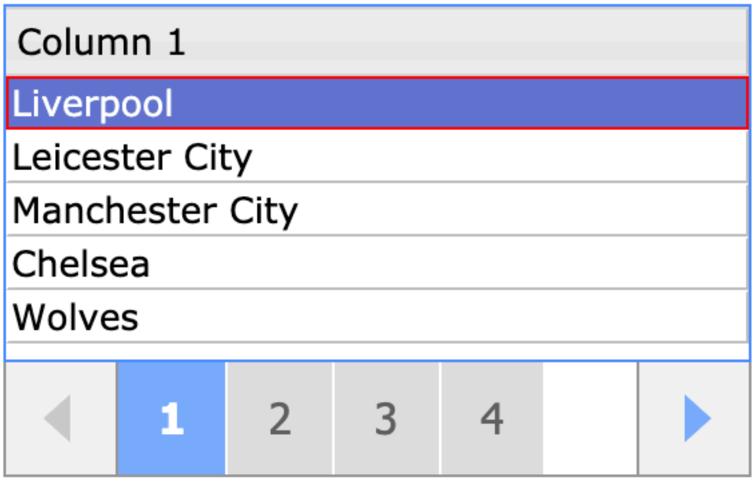
For right justified text (using float so that sort indicators still appear):

```
<div style="float:right;">Param 1</div>
```

You can reassign the column name to force a call to recalculate the HTML for the column header, even if the text has not changed.

List Pager

The Data Grid control has the $pagesize property that allows you to display the lines in the grid as separate pages: see the List Pager section in the List Control for more details.



Column 1
Liverpool
Leicester City
Manchester City
Chelsea
Wolves

Figure 157:

Pick List

You can add a dropdown list or “picklist” to one of the columns in your data grid to allow the end user to select a value from a list of preset values. You must create another list containing the list of preset values and specify its name in the $columnpicklist property for the column in which you want to add the picklist. The datatype of the column in your main data list variable corresponding to the picklist column must be an integer, and $columnmode for the grid column itself must be set to kJSDataGridModeDropList. The integer value stored in this column will correspond to the line number of the selected line in the picklist.

For example, if you want to list a product in your main grid field that has only four possible colors, you could create a sublist containing those color values and assign this sublist to a column in your grid field corresponding to an integer column in your main data list: this would allow the end user to choose a color from a preset list of colors.

The $getpicklist() client method allows picklist columns to be specified dynamically. It is called for every row in the main list for the data grid, allowing you to return a custom list for each row in a Data Grid, if required. It contains three three parameters: pHorzCell, pVertCell and pDataColumnName to assist with generating the required list. The return should be a single column list, with each row being an option in the picklist. In the case of multiple column lists returned, it will only use the first column. Note that you must still specify an instance variable list in $columnpicklist, otherwise the column will not be set up as a picklist type column.

Row Styles

The $rowcsscol property allows you to specify CSS styles for a row in a data grid. The $rowcsscol property specifies the column number in the $dataname list for specifying custom CSS class names to apply to individual rows. Multiple class names can be assigned with a space separated list.

The CSS rules for classes can be added to user.css: it may be necessary to use !important to override existing styles. For example, in user.css:

```
.omnis-datagrid .highlight {
  background: red !important;
  color: white !important;
}
```

Horizontal Padding

The **$horzpadding** and **$columnhorzpadding** properties allow you to set the horizontal padding for all the cells in the grid, or for individual user-defined columns. When $userdefined is kFalse for all columns in the grid, the value of $horzpadding is applied to every cell in the grid, including the grid title, data cells, column header cells, and footer row cells.

When $userdefined is kTrue for a column, the value in $columnhorzpadding is applied to the relevant datas cells, header cells, and footer cells for that column.

Both properties default to 2 for existing data grids in converted libraries to minimize appearance changes. While for new data grids, both properties default to 14 to match the horizontal padding for Edit fields.

Grid Line Visibility

The **$gridlinesvisible** property allows you to select which parts of a data grid will display grid lines; in previous versions you could turn all lines on or off. When using the Property Manager to change the $gridlinesvisible property, a checklist is displayed allowing you to check or uncheck the kJSDataGridVisibleGridLines… constants (see below) to specify which individual lines you want to be displayed.

Constants	Description
kJSDataGridVisibleGridLinesCellHorz	Horizontal cell grid lines
kJSDataGridVisibleGridLinesCellVert	Vertical cell grid lines
kJSDataGridVisibleGridLinesHeader	Header grid line
kJSDataGridVisibleGridLinesColumnHeaderHorz	Horizontal column header grid lines
kJSDataGridVisibleGridLinesColumnHeaderVert	Vertical column header grid lines
kJSDataGridVisibleGridLinesFilter	Filter grid line
kJSDataGridVisibleGridLinesFooterHorz	Horizontal footer grid lines
kJSDataGridVisibleGridLinesFooterVert	Vertical footer grid lines

To set this property in your code, you can add the constant values together to get the desired result, for example:

```
Calculate $cinst.$objs.DataGrid.$gridlinesvisible as kJSDataGridVisibleGridLinesHeader + kJSDataGridVisibleGri
```

For existing grids, those with $gridlinesvisible set to kTrue will have all values selected, and those set to kFalse will have no values selected, which means existing grids should see no change in appearance.

Formatting cells

The $formatcell() client method is fired whenever the selection state of a row in a data grid changes. A new boolean parameter, pSelected, has been added to allow you to style cell values depending on whether the line is selected or not.

Column Data Type Formatting

When you set $columnmode to kJSDataGridModeFormatted, the mode acts like kJSDataGridModeAuto, in that the data grid automatically handles the data based on its type. However, the grid formats the data using the properties $columndateformat, $columndateformatcustom, and $columnnumberformat, rather than the $js...format... properties.

You can use an integer column data type to represent a checkbox. To do this set $columnmode to kJSDataGridModeFormatted and set the $columnnumberformat to "bool". This will cause integer data to be treated as Boolean, where non-zero means true, and zero means false. If the end user updates the grid using the check box, 1 will be stored in the list for true, and zero for false.

Highlighting Cells

The properties $hilitefocusedcell and $cellhilitecolor allow you to highlight the cell that has the focus.

- **$hilitefocusedcell**
  If true, the focused cell will be outlined in the color specified by $cellhilitecolor
- **$cellhilitecolor**
  The color of the focused cell's outline, provided $hilitefocusedcell is kTrue

Grid Scrolling

JavaScript Data Grids have $vscroll and $hscroll properties which allow you to scroll a grid vertically or horizontally at runtime in the client browser; note these properties are write-only meaning that you cannot return their values at runtime.

The vertical scroll value assigned using $vscroll is the position of the scroll bar according to row number in the control. The horizontal scroll value assigned using $hscroll is the designed grid column number for a data grid.

Tabbing through cells

The property $tabthroughcells allows you to change the action of the tab key while the focus is on a data grid. If set to kTrue (default is kFalse), tabbing from a cell which is not being edited selects the next cell, or Shift+tab selects the previous cell. In addition, setting $hcell or $vcell triggers edit mode if $autoedit=kTrue.

Auto Correction and Capitalization

Datagrids have the **$autocorrect** and **$autocapitalize** properties which means that text entered into the cells of a data grid are corrected for spelling and capitalization automatically.

Validating data

The $validate method allows you to validate the data entered into any cell in the grid. If present, the method is called when an edit is made to a grid cell, with the parameters pRow, pCol, pNewValue being passed to the method. The method returns true to indicate that the change is valid, depending on the validation code you add to the method, otherwise the value in the cell will revert to the previous value.

Numeric Data Validation

Data entered in grid columns with the numeric data type will be validated automatically, as it is entered. When the end user tries to enter invalid data into the grid column field, such as an alphabetic character, the data is rejected, and the field is highlighted momentarily to indicate an error (the default action is to show a red border).

When leaving the entry field, the value is normalized, that is, integer data is constrained to the valid range or for other numbers it is rounded to the correct number of decimal places; also, leading zeroes are removed, and so on.

Initial Row Values

When a Data grid has $enterable & $extendable enabled, the user can add a new row by entering data into the empty 'extendable' row at the bottom, and the remainder of the columns in that row are given default values.

However, if you want to override these defaults, you can now implement a method named $initextendrow on the data grid control. This method should return a row with column values set to the appropriate defaults you wish to use. The order and the data type of the columns must match the order and types of the columns of the list defining the Datagrid and specified in $dataname.

Inserting Dates as null values

Data Grid columns have the property $columnallownulldateinput to allow a null value to be added to a row of data when the end user tabs out of the last line of the grid to create a new line automatically.

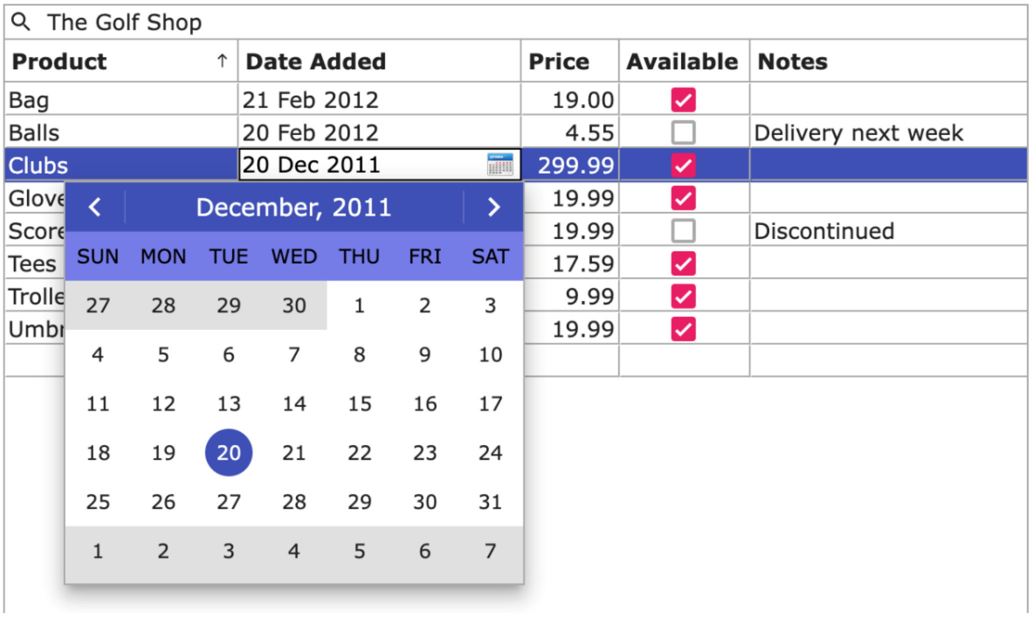
If $columnallownulldateinput is true, and the datatype of the column is Date, cells in the column will default to a value of null when added through the UI. Additionally, if this property is enabled, the end user can change a date to be null by pressing Backspace or Delete while the cell has focus.

If false (the default), the behaviour is unchanged from previous versions. Note it is not possible for the end user to input null values into the grid, via the popup date picker, for example.

Using the Date Picker

The Data Grid uses the appropriate Date Picker according to the constant specified in $dateformat or $columndateformat. If this is set to kJSFormatCustom, then $dateformatcustom or $customdateformatcustom is used as above. If set to kJSFormatNone, then it will attempt to use the data subtype applied to the dataname of the column to determine which picker to use.

Data Grids have the $datepickermode and $datepickerpopupstyle properties, as well as $columndatepickermode and $column-datepickerpopupstyle. The latter two work in the same way, but on the given column when $userdefined = true. The following shows $datepickermode set to kJSDatePickerModeCalendar.



The Golf Shop

Product ↑	Date Added	Price	Available	Notes
Bag	21 Feb 2012	19.00	☑	
Balls	20 Feb 2012	4.55	□	Delivery next week
Clubs	20 Dec 2011	299.99	☑	
Glove		19.99	☑	
Score		19.99	□	Discontinued
Tees		17.59	☑	
Trolle		9.99	☑	
Umbr		19.99	☑	

Figure 158:

Entering Dates Manually

The properties $editdatetext and $columnallownulldateinput allow end users to enter a date manually via the keyboard rather than using the date picker. When set to true, $editdatetext (and $columneditdatetext when $userdefined=kTrue), allows keyboard entry of a date/time. If a date that cannot be parsed is entered, it will revert to the previously stored date, unless $columnallownulldateinput=kTrue, in which case the field data will become null.

Note this has no effect on the date picker popup control, so if you don't want to use the picker you need to apply the following css rule to hide the picker:

```
.datetimepopup-button {
  visibility: hidden;
}
```

Fixed Columns

The $frozencolumns property allows you to fix or "freeze" a number of columns to the left of the grid, so they do not scroll when the other columns in the grid are scrolled horizontally. The property takes a number value from 1 upwards corresponding to the first column on the left of the grid. For example, you could specify a value of 1 to create row headings that are fixed to the left of the grid.

Column Header Justification

The following properties allow you to justify content in data grid column headers.

- **$headerjst**
  A kJSDataGridJst... constant that sets the alignment of the data grid header
- **$columnheadersjst**
  A kJSDataGridJst... constant that sets the alignment of all the column headers; overrides $columnheaderjst
- **$columnheaderjst**
  A kJSDataGridJst... constant that sets the alignment of all the current column's header; $columnheadersjst must be set to kJSDataGridJstDefault

Color Picker

The **kJSDataGridModeColorPicker** column type displays a color picker in the data grid cell allowing the end user to select a color. The end user can click into the color palette on the picker to select a color, or the entry field accepts colors in the hex (the default), rgb or color name formats. You can navigate the color picker from the keyboard without the picker losing the focus.

A numeric color value is returned from the color picker, or a color function can be used to set the color of the column, such as truergb(kDarkGreen), or rgb(255,0,0).

For example, to set the colors for the first 3 lines in the second column, use the code:

```
Do iList.$add('Bag',truergb(kDarkGreen),'21/02/12','19.00',kTrue,'')
Do iList.$add('Balls',rgb(255,0,0),'20/02/12','4.55',kFalse,'Delivery next week')
Do iList.$add('Clubs',rgb(0,0,255),'20/12/11','299.99',kTrue,'')
```

You can specify the text for the OK and Cancel buttons on the color picker using $colorpickeroktext and $colorpickercanceltext.

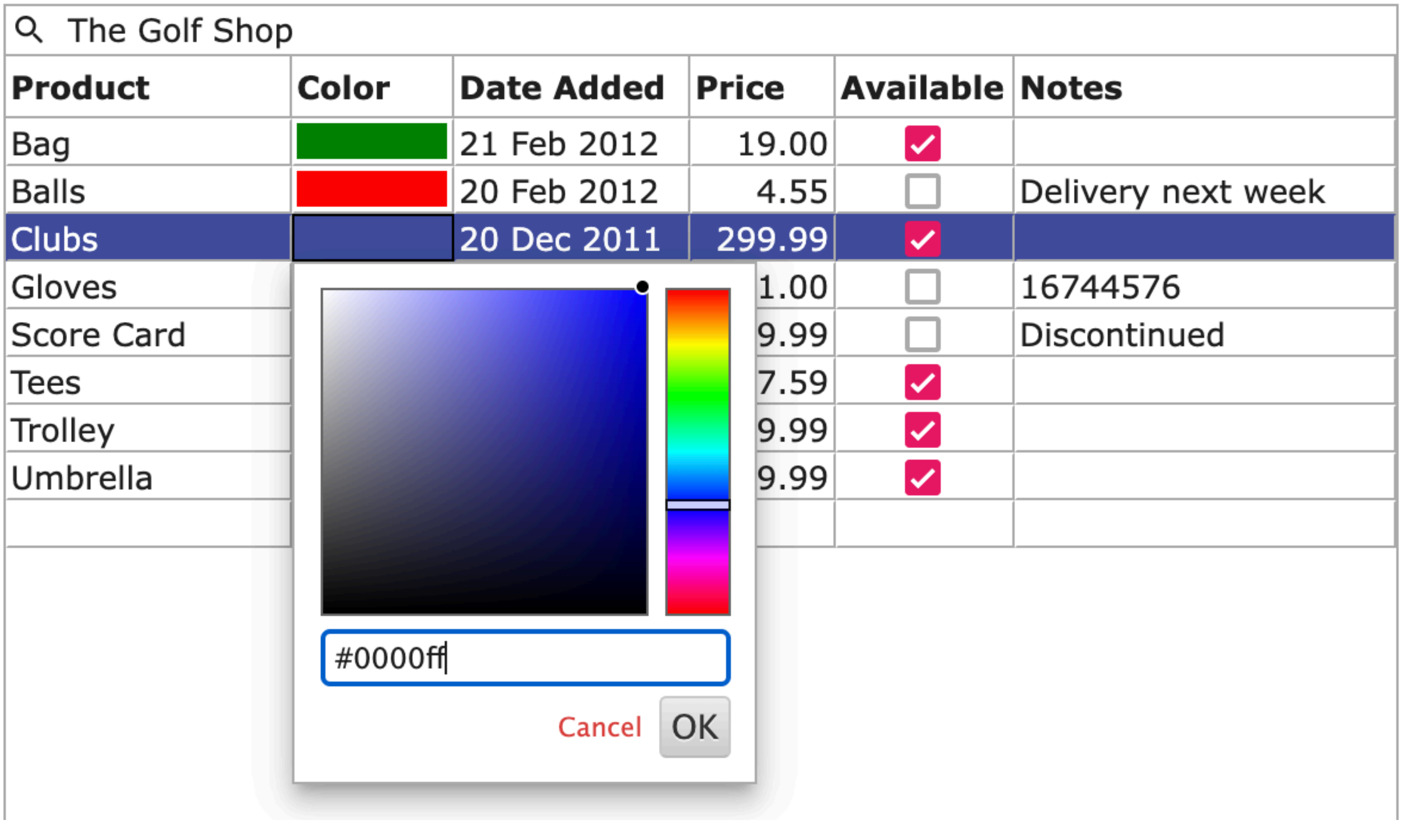
You can localize the strings for the color entry field for the aria-label and aria-describedby accessibility properties, "ctl_dgrd_color_input" and "ctl_dgrd_color_input_desc" respectively.

Number Columns

Data grid columns with type Number have the property $columnzeroshownempty which specifies that values of zero are shown empty rather than displaying a 0 digit.

Hiding a column

The $columnhidden property allows you to hide the specified column at runtime. The default is false, meaning the column is visible.

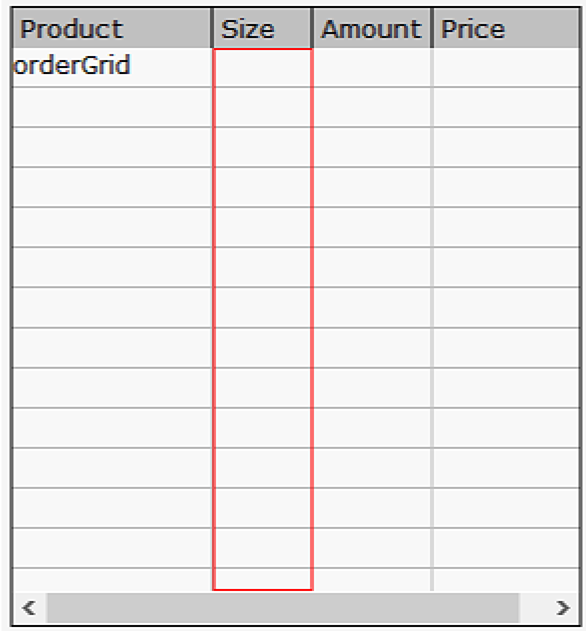


The Golf Shop

Product	Color	Date Added	Price	Available	Notes
Bag	🟩	21 Feb 2012	19.00	☑	
Balls	🟥	20 Feb 2012	4.55	□	Delivery next week
Clubs		20 Dec 2011	299.99	☑	
Gloves			1.00	□	16744576
Score Card			9.99	□	Discontinued
Tees			7.59	☑	
Trolley			9.99	☑	
Umbrella			9.99	☑	

#0000ff

Figure 159:



Product	Size	Amount	Price
orderGrid			

Figure 160:

Data Grid Example

The **Webshop** sample app, available under the **Applets** section in the **Hub,** uses a data grid to display a list of products that have been ordered in the main product jsShop remote form. The data grid control is called ‘orderGrid’ and is seen here in design mode:

The $dataname of the data grid is set to iOrderList which is defined from a table class T_qOrders which is linked to a query class qOrders. When the product form is opened, the $construct method behind the data grid defines the list from the table class.

```
# $construct behind the data grid
Do iOrderList.$definefromsqlclass($tables.T_qOrders)
```

When the end user clicks the ‘Order Now’ button in the product window, the data for the selected product and size/type is passed to the process_order method (as value 1, 2, or 3), which inserts the data into the list (after a check to see if the user has already ordered the same product) and the list is redrawn. The process_order method is as follows:

```
# process_order class method in the jsShop form
# contains pButtonNumber parameter (Short Int) to receive the value of the product button clicked
If iProductList.product_price_[pButtonNumber]>0     ## price must be greater zero
  Do iOrderList.$search(    $ref.order_product_id=iProductList.product_id    &$ref.order_size=iProductList.proc
  If iOrderList.$line   ## found one so increment existing order
    Calculate iOrderList.order_amount as iOrderList.order_amount+1
  Else     ## new one so add to iOrderList
    Do iOrderList.$add( #NULL,#D,iProductList.product_id,iProductList.product_name,      iProductList.product_
    Do iOrderList.$line.$assign(iOrderList.$linecount())
  End If
  Calculate iOrderList.total_price as     iProductList.product_price_[pButtonNumber]*iOrderList.order_amount
  Do $cinst.$objs.checkOutBtn.$enabled.$assign(iOrderList.$linecount()>0)
  Do $cinst.$objs.orderGrid.$redraw()
Else
  Do $cinst.$clientcommand('yesnomessage',row(con('Would you like to order >',iProductList.product_size_1,'< i
End If
```

The Apps Gallery on the Omnis website has a further example showing how you can use the Data grid component.

Date Picker Control

Group	Icon	Name	Description
Other	▦	Date Picker	Date picker with touch selection

The **Date Picker Control** allows the end user to select a single date, a date range, and/or a time, rather than having to enter a date or time from the keyboard; in this case, the UI is better and issues with formatting a date or time are avoided. There is a sample app called **JS Date Picker** in the **Hub** in the **Studio Browser,** and the same app is available in the JavaScript Component Gallery.

You can assign a Date/Time instance variable to the $dataname property to load the date/time selected by the user, or you can assign a two column instance row variable to contain the date/time and time zone offset of the client in the respective columns (see below for info on the time zone offset).

You can assign an empty string to a Date variable on the client, in which case it will be treated as an ‘empty’ date. Assigning 0 to a Date variable on the client sets it to 31 Dec 1900, the same as on the Omnis Server.

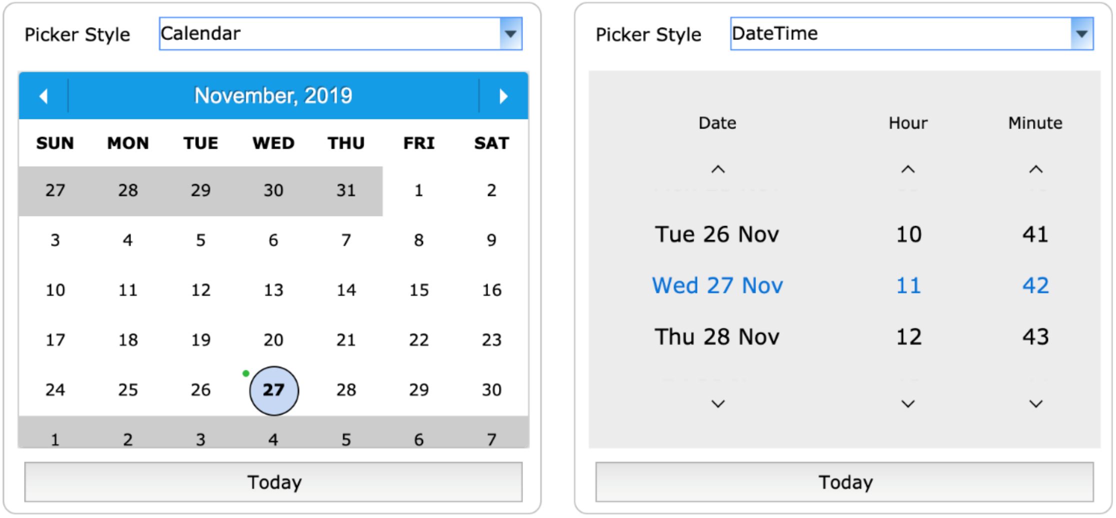
Date Picker Style

The $datestyle property specifies the style or date/time content of the date picker control, which can be a combination of date & time, date only, time only, or a calendar view, as specified by a kJSDatePickerStyle... constant:

Constant	Description
kJSDatePickerStyleDate	a date display only
kJSDatePickerStyleDateTime	a date and time are displayed (right below)
kJSDatePickerStyleTime	a time display only
kJSDatePickerStyleCalendar	a calendar is displayed (left)
kJSDatePickerStyleCustom	a custom format, see below

The color of the Date Picker is specified with $datefacecolor while $datefacealpha sets the transparency (value 0-255).

There is an example app in the **Samples** section in the **Hub** in the Studio Browser showing how you can use the Date picker to allow the end user to select a date; in addition, the **Holidays** example app under the **Applets** option in the Hub uses the Date picker, which is described later in this section.



Picker Style: Calendar

Picker Style: DateTime

Figure 161:

Mode & Popup Style

In addition to the $datestyle property, you can use the $datepickermode and $datepickerpopupstyle properties to control the mode and popup style of the date picker (also applies to data grid cells and columns).

- **$datepickermode**
  controls the type of picker to be displayed, one of the following constants:
  kJSDatePickerModeAuto: Date picker type is assigned automatically based on $dateformat
  kJSDatePickerModeCalendar: calendar type is displayed
  kJSDatePickerModePicker: a picker type is displayed

- **$datepickerpopupstyle**
  controls how the popup is displayed, one of the following constants:
  kJSDatePickerPopupStyleAuto: Popup style is determined by device type
  kJSDatePickerPopupStyleInline: Popup style will always be displayed adjacent to the control
  kJSDatePickerPopupStyleModal: Popup style will always be displayed modal

(Note that Internet Explorer does not correctly display the modal type, and so falls back to inline on these clients.)

The inline style picker will position itself underneath the parent control, but from the right so it is closer to the icon which opens it. If there is not enough space beneath the parent control, the picker will be placed above, where space permits.

Calendar Style Picker

The Date Picker control can be switched to display a calendar style date picker, by setting $datestyle to kJSDatePickerStyleCalendar. There are a number of properties that apply to the control when the date style is set to calendar.

Picker style on mobile devices

When $datestyle is set to kJSDatePickerStyleCalendar desktop browsers will display the calendar as expected. On mobile devices however, even when $datestyle is set to kJSDatePickerStyleCalendar, a calendar will be replaced with a date picker (same as kJSDatePickerStyleDate), since a picker style date selector is the preferred style on mobile devices. This can be overridden by setting the property $datestyleusepickeronmobile to kFalse.

Start and End dates

The following properties allow you to set the start and end dates (minimum and maximum):

Property	Description
$mindate	Only assignable at runtime, this is a Date to set the start date (minimum selectable limit on the calendar)
$maxdate	Only assignable at runtime, this is a Date to set the end date (maximum selectable limit on the calendar)

To allow the same functionality in a popup date picker (in the Data grid or Edit field) you can use the $getdisableddates() method, which defaults to client-executed, or it can also run on the server if required. This method must return a Row containing up to 4 columns, which should be named the same as the relevant properties which set disabled dates above but without the $, that is, datesdisabled, daysofweekdisabled, maxdate, mindate. They can be in any order, and not all need to be included. Their data type should be same as the properties above, apart from datesdisabled, which should be a list of dates (i.e. not just an instance variable name).

Week Number

You can display the week number in the calendar view of the Date Picker control by enabling the $showweeknumber property and setting the associated color properties. When set to kTrue, the $showweeknumber property displays the iso week number on the left side of the calendar style date picker (when $datestyle is set to kJSDatePickerStyleCalendar). The $weeknumbertextcolor property specifies the text color of the week numbers, and $weeknumbercolor controls the background color of the week number area.

Disabling Dates

The calendar style **Date Picker** allows you to disable specific dates using the following properties:

Property	Description
$datesdisabled	an instance variable containing a list with a single column of type Date. This is to disable individual dates on the calendar

Property	Description
$daysofweekdisabled	an integer made up from flags to specify days of the week to disable (e.g. you might want to disable Saturdays and Sundays). This is presented as a check list in the Property Manager, but to assign via code there are new constants to use, kJSWeekDaySun through to kJSWeekDaySat. Assign kJSWeekDayNone (resolves to 0) to set this property to no disabled days
$disableddaycolor	The color used for disabled days. This defaults to kDefault so that it just inherits the $daycolor or $otherdaycolor
$disableddaytextcolor	The text color used for disabled days. This defaults to kJSThemeColorDisabledText

In addition, disabled days have a strikethrough text appearance (equivalent to the line-through css attribute).

Custom Date Style

The $datestylecustom property can be used in conjunction with setting $datestyle to kJSDatePickerStyleCustom to set the style or format of the date shown. You can enter a string of characters to represent the columns required as per the Omnis date/time format strings, for example, “mdy” to specify Month, Day, Year columns in that order.

In addition, you can specify a grouped column by enclosing the date characters in parenthesis, for example, “(wdm)” will specify a single coumn containing Weekday, Day, Month. Note: this column will always alter the day by one by increasing or decreasing it, so it only makes sense to use this type of column if it includes a day or weekday. Time elements entered into a grouped column will be ignored. Repeated characters are ignored and only one group can be used (further groups are ignored). Groups take precedence over individual columns, therefore “d(wdm)y” will be treated as “(wdm)y”.

Date pickers (other than custom) pick up the locale of the client and display the picker in their standard format. For example, the date picker will display Day, Month, Year in the UK, and Month, Day, Year in the USA (assuming their location settings are set correctly).

Selecting a Date Range

When specified, the properties $rangeselection and $rangeenddataname allow the end user to select a date range, that is, a start date and an end date. The first being a boolean to put the calendar into range selection mode. When true, the end user can select a range of dates by selecting one date after another. The $rangeenddataname property is the name of an instance variable to store the end of the data range and should be of type Date. The variable in $dataname will always hold the start date in range selection mode.

A boolean parameter, pInRangeSelection, will be passed as true with evDateClick when the end user has selected the first date, and false once they have selected the second. If $rangeselection is kFalse, this parameter is not passed, and therefore will return NULL if tested on evDateClick.

The evDateRangeChange fires every time a date range selection has been completed (and $rangeselection is kTrue). This passes two parameters: pStartDate and pEndDate. This means you can obtain a date range without using instance variables if you just need to react to the date range selected. evDateChange does not fire when $rangeselection is kTrue.

The $currdaycolor property applies to inside the current day indicator ring and not the whole cell. This ensures the type of cell is still understood by the end user. E.g. When $todayscolor is different to $daycolor, the end user can still see that it is today, even when they have selected it as the current day.

Events

The following events are generated when the end user clicks on a date picker and/or selects a date: you can create an event handling method on the control allowing you to load the selected date.

- **evDateChange**
  Sent to the control when the current date is changed (not fired when $rangeselection is kTrue); see below
- **evDateClick**
  Sent to the control when the user clicks on a date; only applicable to Calendar type date pickers.
- **evDateDClick**
  Sent to the control when the end user double clicks on a date (not fired when $rangeselection is kTrue)
- **evDateRangeChange**
  Sent to the control when a date range has changed (only fired when $rangeselection is kTrue)
- **evCalendarViewChanged**
  Sent to the control when the view changes in the calendar mode of the date picker; see below

Time Zone Offset

You can return the time zone offset of a date value when it is passed back from the client by using a two column row variable in $dataname; the first column should be defined as a Date Time variable and the second of type Number. If the server passes a new value to the client, then only the first column is significant and should specify the new date to be sent to the client.

When **evDateChange** signals that there has been a change on the client then the updated date is passed back to the server in the first column of the row variable as a UTC/GMT date value and the time zone offset of that value in the client's current time zone is passed back in the second column. The time zone offset is the number of minutes from UTC/GMT, e.g. GMT+2 the time zone offset is 120. This can be used to calculate the date in the time zone of the client.

If time zone offset information is not required, $dataname can be specified as a Date Time instance variable only.

Calendar View Change Event

The **evCalendarViewChange** event is triggered when the view changes in the calendar mode of the date picker; the parameters will vary depending on the current view:

- **pView**
  will be one of kJSDatePickerCalendarViewDays, kJSDatePickerCalendarViewMonths, kJSDatePickerCalendarViewYears, kJSDatePickerCalendarViewDecades
- **pMonth**
  Integer 1-12 for the current month in view (only populated if pView = kJSDatePickerCalendarViewDays)
- **pYear**
  Integer for current year in view (only populated if pView = kJSDatePickerCalendarViewDays or kJSDatePickerCalendarViewMonths)
- **pStartYear**
  Integer for the first year in view (only populated if pView = kJSDatePickerCalendarViewYears or kJSDatePickerCalendarViewDecades)
- **pEndYear**
  Integer for the last year in view (only populated if pView = kJSDatePickerCalendarViewYears or kJSDatePickerCalendarViewDecades)

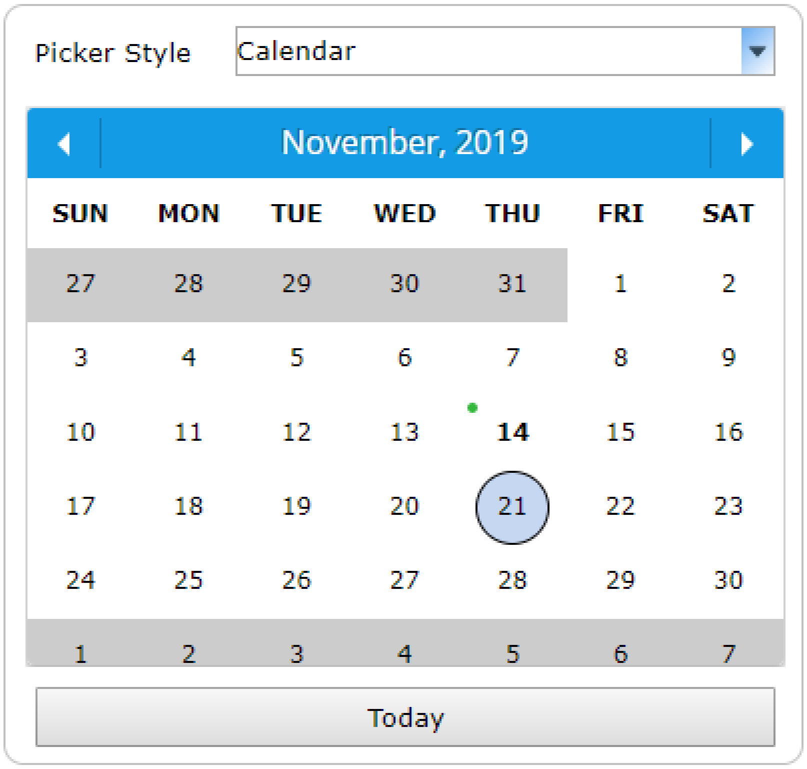
Example

In the **Holidays** sample app uses the Date picker control set to kJSDatePickerStyleCalendar to allow users to select the dates for the holiday applications. The jsUserForm in the Holidays app has two buttons to allow the end user to select a "From" date or "To" date to specify the Begin and End dates for their holiday request. The "From" button has the following code:

```
On evClick
  Calculate iUsingCalendar as kTrue
  Calculate iSelectFrom as kTrue  ## this is the From button
  Do method openCalendar
```

The openCalendar class method moves the calendarPane into view on the main form and has the following code:

```
Do method enableFields (kFalse) ## enables the calendar pane & disables the name pane
Do $cinst.$objs.calendarPane.$top.$assign(90)
Do $cinst.$objs.calendarPane.$left.$assign(150)
Do $cinst.$objs.calendarPane.$visible.$assign(kTrue)
```


Picker Style: Calendar

SUN	MON	TUE	WED	THU	FRI	SAT
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

Figure 162:

If you examine the Holidays app to look at the Date picker note that it is on a page pane field called calendarPane located on the jsUserForm. The $left property for the calendarPane is set to 990, to hide it from view, therefore you'll need to select it using the Field List (right-click the remote form, select Field List and check the calendarPane) and set its $left property to 200 in the Property Manager in order to see it.

The $dataname of the Date picker control itself is set to iCalendarDate, an instance variable of type Date Time and subtype 'D m y'; when the end user selects a date this variable is set to the selected date automatically.

The Date picker has a simple event method to detect when the end user double-clicks on a date cell; it has the following code:

```
On evDateDClick
  Do method closeCalendar
```

The closeCalendar class method passes the date from iCalendarDate into either the iFromDate or iToDate variable defined in the form; it has the following code:

```
If iSelectFrom  ## if the From button
  Calculate iFromDate as iCalendarDate
Else    ## it must be the To button
  Calculate iToDate as iCalendarDate
End If
Do $cinst.$objs.calendarPane.$left.$assign(1250)
Do method enableFields (kTrue)
```

The final two lines of the method move the calendarPane off to the right and enables the fields on the Name pane.

Date Picker Localization

The following strings are available in the JS localization string table to allow you to localize strings for the Date Picker. Note that some of the strings are now arrays of strings to simplify localization (e.g. for months, days of the week, etc).

Date picker specific strings

The following are specific to the date picker control:

```
"ctrl_date_increase": "Increase"
"ctrl_date_decrease": "Decrease,
"ctrl_date_time_button": "Open time picker"
"ctrl_date_calendar_button": "Open date picker"
"ctrl_date_header": ["Select a Month", "Select a Year", "Select a Decade", "Select a Time"]
```

Generic strings

The following strings are available for controls that refer to dates, including the date picker:

```
"month_names": ["January", "February", "March", "April", "May", "June", "July", "August",
"September", "October", "November", "December"]
"month_names_short": ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct",
"Nov", "Dec"]
"day_names": ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"]
"day_names_short": ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
"date_units": ["Day", "Month", "Year", "Decade"]
"time_units": ["Hour", "Minute", "Second", "Millisecond"]
```

The following example applies Spanish text to the Calendar:

```
<script type="text/javascript">
  jOmnisStrings.es = {
    "month_names":      ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto", "Septiembre
    "month_names_short":     ["enero", "feb.", "marzo", "abr.", "mayo", "jun.", "jul.", "agosto", "sept.", "oc
    "day_names":        ["Domingo", "Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado"],
    "day_names_short":      ["DOM", "LUN", "MAR", "MIÉ", "JUE", "VIE", "SÁB"],
    "date_units":     ["Día", "Mes", "Año", "Década"],
    "time_units":        ["Horas", "Minutos", "Segundos"],
    "ctrl_date_header":      ["Selecciona un mes", "Seleccione un año", "Seleccione un década", "Seleccione un
};
</script>
```

See the Localization chapter for more information about setting the strings in the jOmnisStrings object.

Device Control

Group	Icon	Name	Description
Other	▯	Device	Allows access to hardware and ser a mobile device using the JS wrap

The **Device Control** allows you to access features on a mobile device, such as getting the location of the end user's device using GPS, retrieving the contacts information from the device, or returning images from either the camera or photo library on the device. Depending on the type of application you are creating, some or all these features may be useful to enhance the interactivity and functionality of your app for end users when they run your app on a mobile device, such as a phone or tablet.

The *Device control itself is invisible* and to enable access to the device functionality you need to add the Device Control to your remote form and assign an action to the $action property of the control at runtime using methods.

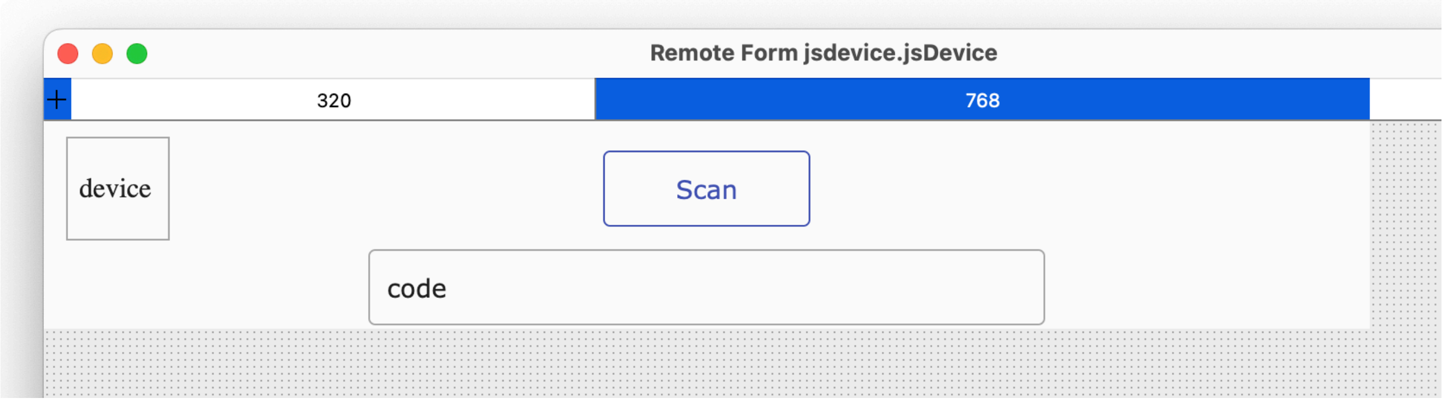


Figure 163:

Running the Device Control and Compatibility

Important Note: *in all but a few specific cases, the actions enabled via the Device Control only work in an application that is running inside one of the JavaScript Wrappers (or Omnis App Manager) on a mobile device*. The exceptions to this are the **Email, Call,** and **SMS** actions which will attempt to work if the app is running in a standard web browser, but it's not guaranteed the actions will always work as expected.

Therefore, you will need to compile your app as a standalone mobile app using the JavaScript Wrappers (for iOS or Android), and test your app in a simulator or directly on a mobile device by running the native app, for the majority of the actions in the Device control to work. Alternatively, on iOS you can use the Omnis App Manager to test your mobile app and the functions of the Device Control, prior to compiling it into a standalone app using the iOS wrapper.

There is a Tech note describing how you can use the Device control and a device's camera to return the info from a barcode or QR code: see TNJC0013 "Reading a barcode or QR code in a mobile app"; this demonstrates how you use the Device control and the **Omnis App Manager** to test your app.

Testing Hardware Features

The Device control supports several hardware functions, some of which may not be available on specific devices or mobile operating systems. You should test your app thoroughly on the specific devices you wish to support with each of the device functions that you want end users to access.

For some of the hardware features, Omnis can detect if they are not present on the current mobile device running the app. For example, if a device does not have a hardware camera, then the action kJSDeviceActionTakePhoto will report an evPhotoFailed message.

Properties

Note the Device control is invisible, therefore some of the visual properties normally associated with JavaScript components may not be relevant, such as $alpha; the following properties are available.

Property	Description
$action	The "action" for the Device control which specifies which function on the client mobile device is accessed; this is assigned as a kJSDevice… constant: see below
$communicationaddress	Character value determining the phone number when the Make Call and Send SMS device actions are used (for SMS only, can be a list of phone numbers), or email address when the Send Email device action is used. Can only be assigned at runtime.

Property	Description
$communicationdata	Character data to be sent as the message body when the Send SMS or Send Email device actions are used.Can only be assigned at runtime.
$communicationsubject	Character data to be sent as the subject when the Send Email device action is used.Can only be assigned at runtime.
$dataname	The name of a List instance variable for the Device control. It will be populated with contact details when using the Get Contacts device action.
$deviceimage	Contains a Character/Binary instance variable name used for holding the image returned from the device. The image will be in base64 format.
$imageaspect	Sets the aspect ratio of the image, a floating number, indicating *width* divided by *height*. A value of 0 no aspect ratio will be enforced, a value of 1 enforces a square image
$imagejpeglevel	The JPEG quality of images returned (0-100). 100 being max quality, 0 being max compression.
$imagemegapixel	A float value indicating the maximum Megapixel resolution of images returned. 0 means no limit.
$imagesizemenu	If true, a dialog will be opened when using the device image actions, to allow the user to pick a size for the image, respecting $imagemegapixel.
$soundname	Name of the sound sample to be played when the Beep device action is called.
$contact…	The $contact… properties determine whether particular pieces of contact information are returned when using the Get Contacts device action, e.g. disabling $contactphotos can significantly reduce the time taken to fetch contacts.

Contact properties

There are several properties that are only relevant when the device action is set to kJSDeviceActionGetContacts which allows you to obtain information from the contacts database on the device. By setting these properties you can control what information is returned from the Contacts data on the device. For example, if $contactname is set to kTrue the contact request will include name info, and so on.

Device Action Properties

The following list summarizes the actions available, and the constant needed for the $action property:

- **kJSDeviceActionBeep** – Beep Device
  forces the device to play the default beep
- **kJSDeviceActionGetBarcode** – Get a Barcode or QR code
  returns the output from Barcode/QR-code scanning function on the device (if available); the output is usually a string which can be a URL
- **kJSDeviceActionGetContacts** – Get contact info
  returns contact information from the device; note there are other properties to determine the content or extent of the contact information returned

- **kJSDeviceActionGetGps** – Get the device location
  returns the location data using the GPS function on the device

- **kJSDeviceActionGetImage** – Get an Image
  returns an image from the device’s image gallery

- **kJSDeviceActionGetUniqueID** – Get Unique ID
  returns the unique ID of the device

- **kJSDeviceActionMakeCall** – Make a Phone Call
  forces the device to make a phone call (if available)

- **kJSDeviceActionSendEmail | kJSDeviceActionSendSms** – Send an Email or SMS
  forces the device to send an Email or SMS / text message (if available)

- **kJSDeviceActionTakePhoto** – Take a Photo
  forces the device to take a photo (if a camera is available)

- **kJSDeviceActionVibrate** – Vibrate the Device
  forces the device to vibrate (if available)

The basic method to assign an action to the Device control is as follows:

```
Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceAction…)
# where oDevice is the name of the Device control
```

Events

Event	Description
evBarcodeFailed	Sent when no Barcode could be obtained from the device.**Parameters**pEventCode - The event code
evBarcodeReturned	Sent to the device control when a Barcode is ready for processing.**Parameters**pDeviceBarcode - The Barcode datapEventCode - The event code
evContactsFailed	Sent when no contacts could be obtained from the device.**Parameters**pEventCode - The event code
evContactsReturned	Sent to the device control when contacts info is ready for processing.**Parameters**pEventCode - The event code
evGpsReturned	Sent to the device control when Location Data is ready for processing.**Parameters**pDeviceGps - The GPS locationpEventCode - The event code
evImageFailed	Sent when the device failed to return an image.**Parameters**pEventCode - The event code
evImageReturned	Sent to the device control when an image is ready for processing.**Parameters**pEventCode - The event code
evPhotoFailed	Sent when the device failed to take a photo.**Parameters**pEventCode - The event code

Event	Description
evPhotoReturned	Sent when an image is returned from camera ready for processing.**Parameters**pEventCode - The event code
evUniqueIDReturned	Sent when a unique ID is returned from the device.**Parameters**pDeviceUniqueID - The unique IDpEventCode - The event code
Standard	evExecuteContextMenu evOpenContextMenu

Beep Device Action

To make the device play a given sound sample, you need to assign the constant kJSDeviceActionBeep to the $action property. This is one-way communication with the device which will result in the device playing a sound sample. To specify which sound to play, you need to set the $soundname property to the name of the sound sample to be played which must be compiled into the wrapper application. The wrapper contains a default sound called “notify”.

Example

```
On evClick
  Calculate $cinst.$objs.oDevice.$soundname as "notify"
  Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionBeep)
```

Get Barcode Device Action

You can return a Barcode or QR code by assigning the constant kJSDeviceActionGetBarcode to the $action property. If the action is successful an evBarcodeReturned event is sent to the Device Control and the barcode data is returned in the pDeviceBarcode event parameter; the barcode data is usually a string containing Alphanumeric characters, such as a product number or name, or in the case of a QR code it could be a website URL.

Example

```
# event method for "Scan" button
On evClick
  Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionGetBarcode)
```

The event method for the Device Control could be:

```
On evBarcodeReturned
  Do iProducts.$search(iProducts.iProdQrCode=pDeviceBarcode,kTrue,kFalse,kFalse,kFalse)
  If iProducts.$line=0
    Do iProducts.$line.$assign(iProducts.$linecount)     ## other
  End If
  Do iProducts.$loadcols()
  Calculate iAmount as 1
  If iProdName='Other'
    Calculate iProdName as pDeviceBarcode     ## show the value of the barcode
  End If
```

There is a Tech note describing how you can return the info from a barcode or QR code using the Device control and the **Omnis App Manager** to test the Device control: see TNJC0013 “Reading a barcode or QR code in a mobile app”.

Vibrate Device Action

To make the device vibrate you need to assign the constant kJSDeviceActionVibrate to the $action property. This is one way communication with the device which will result in the device vibrating for a short period of time.

Example

```
On evClick
  Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionVibrate)
```

Get GPS Device Action

To receive location (GPS) data from the device you need to assign the constant kJSDeviceActionGetGps to the $action property. The evGpsReturned event is sent when the location data has successfully been returned. The event parameter pDeviceGps will contain the returned data which is formatted as a string containing longitude and latitude data separated by a colon ":". If the device fails to obtain location data or the device does not support location tracking, the returned data will be a longitude and latitude of zero, i.e. "0.0:0.0".

Example

```
On evClick
  Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionGetGps)
On evGpsReturned
  Calculate $cinst.$objs.oMap.$latlong as pDeviceGps
```

Take Photo / Get Image Device Action

To take a photo with the device (if a camera is present) or to return an image from the device's gallery the kJSDeviceActionTakePhoto or kJSDeviceActionGetImage constants need to be assigned to the $action property. The $deviceimage property of the Device component needs to be assigned to an Instance Variable of type Binary or Character to hold the incoming image data from the device. If the device is successful in returning an image, the event evPhotoReturned or evImageReturned will be called to indicate that an image was returned, whereupon the instance variable specified in $deviceimage will be populated with the base64 encoded image data. In the event of the device failing to return an image or the user cancels the request, the event evPhotoFailed or evImageFailed will be sent.

Example

```
On evClick
  Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionTakePhoto)
```

In this case the instance variable iImage is a Binary variable. The $deviceimage property is set to iImage. There is another Binary variable called ipic which is associated to a picture component. By copying the returned image in iImage into the picture component variable you can display the image returned from the device.

```
On evPhotoReturned
  Calculate iPic as iImage
```

In addition to the TakePhoto action, the device control has the client-executed method, $takephoto(iWidth, iHeight) which provides a shorthand way of taking a photo with specific dimensions.

Image Aspect Ratio

The $imageaspect allows you to specify the aspect ratio of a photo; it only affects images taken with the TakePhoto device action, not the GetImage action. This functionality is only available in the iOS and Android wrappers - version 3.1.0 & later; also note the minimum Android version is now API21 (5.0, Lollipop).

The $imageaspect property takes a floating number, indicating *width* divided by *height*. If set to 0, no aspect ratio will be enforced, and the standard camera application will be used for taking photos. If greater than zero, a custom camera view within the app will be used, which shows the preview stream in the specified aspect ratio, and an image of the specified aspect will be returned. A value of 1 will enforce a square image.

The $imageaspect property can be used in conjunction with $imagemegapixel to take an image of specific dimensions, that is:

$imageaspect = targetWidth / targetHeight

$imagemegapixel = (targetWidth * targetHeight) / 1,000,000

Get Contacts Device Action

To obtain information from the contacts database on the device the kJSDeviceActionGetContacts constant must be assigned to the $action property. To accommodate the contact database the $dataname property needs to be assigned to an Instance Variable of type List. The properties starting with $contact… determine which contact fields will be obtained from the device: setting these properties to true or false will determine if that specific field is returned from the device.

The evContactsReturned event is triggered when the contact database has been returned, and in the case of the device failing to obtain the contact database the evContactsFailed event is triggered.

Example

```
On evClick
  Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionGetContacts)
# retrieve info from the Contact list
On evContactsReturned
  Set reference lNameRow to iDeviceList.name.1
  Calculate iNameRow.FirstName as lNameRow.givenName
  Calculate iNameRow.MiddleName as lNameRow.middleName
  Calculate iNameRow.Surname as lNameRow.familyName
  Calculate iNameRow.Prefix as lNameRow.honorificPrefix
  Calculate iNameRow.Suffix as lNameRow.honorificSuffix
  Calculate iNameRow.Nickname as iDeviceList.nickName
```

Contacts data structure

- **displayName:** The name of this Contact, suitable for display to end-users (String).
- **name:** A row containing all components of a contact’s name.
  formatted: The complete name of the contact (String).
  familyName: The contact’s family name (String).
  givenName: The contact’s given/first name (String).
  middleName: The contact’s middle name (String).
  honorificPrefix: The contact’s prefix (example Mr. or Dr.) (String).
  honorificSuffix: The contact’s suffix (example Esq.) (String).
- **nickname:** A casual name to address the contact by (String).
- **phoneNumbers:** A list of all the contact’s phone numbers.
  type: A string that tells you what type of phone number this is (example: ‘home’) (String).
  value: The phone number (String).
  pref: Set to true if this is the user’s preferred value (Boolean).

- **emails:** A list of all the contact's email addresses.
  type: A string that tells you what type of email this is (example: 'home') (String).
  value: The email address (String).
  pref: Set to true if this is the user's preferred value (Boolean).
- **addresses:** A list of all the contact's addresses.
  pref: Set to true if this is the user's preferred value (Boolean).
  type: A string that tells you what type of address this is (example: 'home') (String).
  formatted: The full address formatted for display (String).
  streetAddress: The full street address (String).
  locality: The city or locality (String).
  region: The state or region (String).
  postalCode: The zip code or postal code (String).
  country: The country name (String).
- **ims:** A list of all the contact's IM addresses.
  type: A string that tells you what type of IM this is (example: 'home') (String).
  value: The IM username (String).
  pref: Set to true if this is the user's preferred value (Boolean).
- **organizations:** A list of all the contact's organizations.
  pref: Set to true if this is the user's preferred value (Boolean).
  type: A string that tells you what type of organization this is (example: 'work') (String).
  name: The name of the organization (String).
  department: The department the contact works for (String).
  title: The contacts title at the organization (String).
- **birthday:** The birthday of the contact (Character).
- **note:** A note about the contact (String).
- **photos:** A list of the contact's photos. In general, there will be only one row.
  type: A string that tells you what type of field this is (example: 'home') (String).
  value: The photo data, encoded as base64. These are small, thumbnail photos. (String).
  pref: Set to true if this is the user's preferred value (Boolean).
- **categories:** A list of all the contacts user defined categories.
  type: A string that tells you what type of category this is (example: 'home') (String).
  value: The value of the field (such as a phone number or email address) (String).
  pref: Set to true if this is the user's preferred value (Boolean).
- **urls:** A list of web pages associated with the contact.
  type: A string that tells you what type of web page this is
  (example: 'home') (String).
  value: The website URL (String).
  pref: Set to true if this is the user's preferred value (Boolean).

Make a Call Device Action

To make a phone call from the device the kJSDeviceActionMakeCall constant is used. Before assigning this action the phone number for the call should be specified in the $communicationaddress property.

Example

```
Do $cinst.$objs.oDevice.$communicationaddress.$assign("0123456789")
Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionMakeCall)
```

Send an SMS or Email Device Action

To send an SMS (text message) or Email from the device the kJSDeviceActionSendSMS or kJSDeviceActionSendEmail constant is used. Before assigning this action the phone number or list of numbers (for the SMS action) or the email address (for Email) should be specified in the $communicationaddress property. The message body of the SMS or email can be specified in $communicationdata. The subject of the email can be specified in $communicationsubject.

Example

```
Do $cinst.$objs.oDevice.$communicationaddress.$assign("0123456789")
Do $cinst.$objs.oDevice.$communicationdata.$assign("A message")
Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionSendSMS)
```

You can send a SMS to multiple recipients by assigning a comma-separated list of phone numbers to $communicationaddress. For example:

```
Do cinst.$objs.oDevice.$communicationaddress.$assign("0123456789,0987654321,0192837465")
```

Getting the Unique ID Action

You can return the unique ID of the device running your standalone app using the kJSDeviceActionGetUniqueID action.

```
Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionGetUniqueID)
```

The ID is unique to each JS wrapper installation and changes when the app is re-installed.

If successful, the action triggers the evUniqueIDReturned event with pDeviceUniqueID containing the ID. You can use the $event method of the device control to return the ID:

```
On evUniqueIDReturned
    Calculate iDeviceID as pDeviceUniqueID
```

Running Device Actions in the Browser

Some of the Device actions (those listed below) may work in your application when running in a standard desktop web browser, that is, outside a wrapper. However, the results of running any of these actions in a web browser are very unpredictable, mainly due to the great variation among different web browsers and operating systems, therefore we do not recommend or support apps using these actions in a web browser. If you do use them however, you should test these actions thoroughly.

Call, SMS, and Email actions

You can use the **Call, SMS,** and **Email** actions in an application running in a browser, and not in a wrapper, and the web browser on the client will attempt to execute the relevant action (this only applies to these actions: all other actions have to be executed inside the wrapper). For example, if you run the Email action in a web browser it will attempt to initiate an email in the email program on the client.

Vibrate and Location actions

When run outside the wrapper the **Vibrate** action is not currently supported on iOS Safari, and hence other iOS browsers, as they all have to based on Apple's WebKit.

The **Location** action only works over HTTPS in recent browsers.

Droplist Control

Group	Icon	Name	Description
Lists	▤	Droplist	List that drops down when clicked

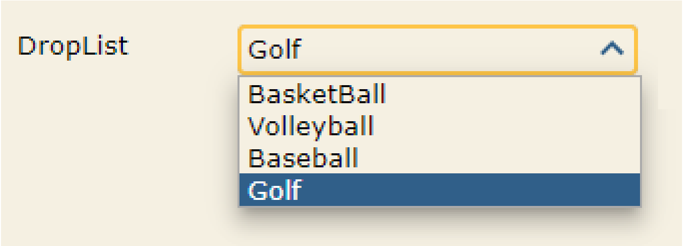


Figure 164:

The **Droplist Control** displays a dropdown list from which the end user can make a selection; the contents of the list can be supplied from a default list or a list variable which can be built dynamically. There is an example app in the **Samples** section in the **Hub** in the Studio Browser (called **JS Droplist, Combo, Popup**), and the same app is available in the JavaScript Component Gallery. The following screen shows the Vintage JS Theme in use.

You can specify a default list of options in the $defaulttext property, which is a comma-separated list of options; $defaultline is the default line (set to 1) which is selected when the form is opened (only when $defaulttext is used). Alternatively, you can assign the name of a list instance variable to $dataname to populate the list; $listcolumn specifies which column of the list variable is used to display the list. The $::listheight property specifies the height of the droplist.

The $seldataname property allows you to specify the name of an instance variable, which will be populated automatically with the selected value from the droplist.

Droplist Style

The Droplist and Combo Box controls have the $dropliststyle property to allow you to apply a rounded style to the list part of the control. The style of the droplist is a kJSDropListStyle... constant:

- **kJSDropListStyleDefault**
  The default droplist style (shown below on Windows)

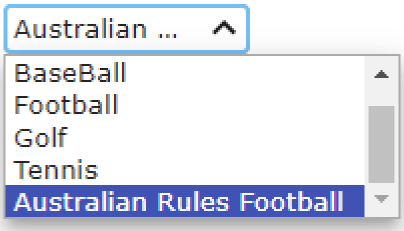


Figure 165:

- **kJSDropListStyleRounded**
  The $borderradius property is applied to the combined field and list part of the control when it is dropped; if the dropped list is wider than the field, its width is temporarily increased to match (shown below on macOS)

Horizontal Padding & Extra space

The Droplist and Combo Box controls have the $horzpadding property to allow you to add extra horizontal padding, in pixels, to the text in the list part of the control. This property has also been added to Combo boxes.

The $extraspace property can be used to add extra space to the line content in the list. If $extraspace is zero, the height of each row is the default height of the row content. If $extraspace is greater than zero, the height of each row is the font height + $extraspace.

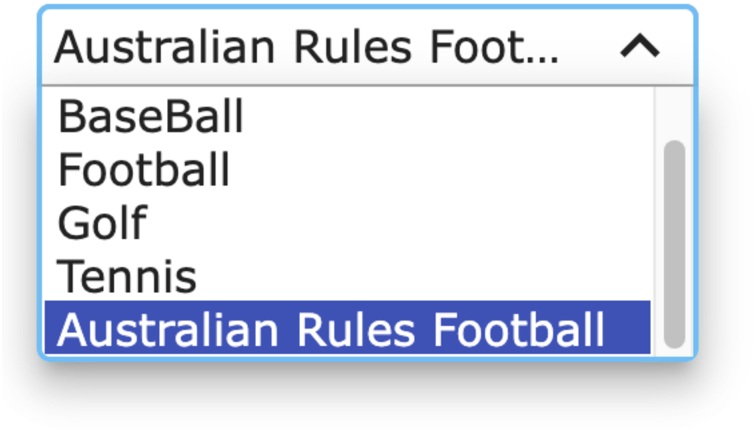


Figure 166:

Selected Line

The $selectonopen property allow you to manage whether or not the first line is selected when a droplist is opened. When $selectonopen is true (the default) and no line has been set, the first line in the droplist will be selected when it is opened, and the evClick event will be sent. The property is set to kTrue for droplists in existing libraries, to maintain behavior as in previous versions, whereby the first line was selected and evClick sent when no line was set. You can set $selectonopen to false to stop the first line in the droplist being selected when the form is opened.

Events

When a line in a Droplist is selected an evClick is generated with the selected list line reported in pLineNumber.

Example

The jsUserForm in the **Holidays** sample app uses a droplist to allow users to select an employee to view their holiday leave requests. The $dataname of the empList Droplist control is set to iEmployeeList which is built via the $construct method when the form is opened.

```
# buildEmpList class method in the jsUserForm
Do iEmployeeList.$definefromsqlclass('sEmployee')   ## the schema
Do iEmployeeList.$sessionobject.$assign(iSQLObjRef)
Do iEmployeeList.$select()
Do iEmployeeList.$fetch(kFetchAll)
Do iEmployeeList.$cols.$add(iEmpFullName)
Calculate lTotal as iEmployeeList.$linecount
For lNum from 1 to iEmployeeList.$linecount step 1
  Calculate iEmpFullName as con(iEmployeeList.[lNum].FirstName,kSp,iEmployeeList.[lNum].LastName)
  Do iEmployeeList.[lNum].$assigncols(,,,,,iEmpFullName)
End For
Do iEmployeeList.$line.$assign(1)
Calculate iName as iEmployeeList.iEmpFullName
```

The droplist control contains a $event method which is triggered when the user selects a line in the list; the code in the event method redraws the holiday list for the selected employee:

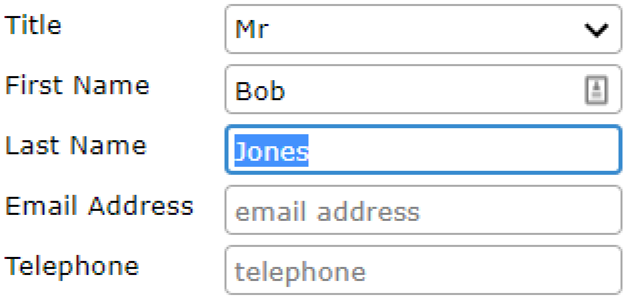
```
On evClick
  Do method buildHolidayList
  Calculate iName as iEmployeeList.[pLineNumber].iEmpFullName
  Do $cinst.$objs.pagePane.$objs.holidayList.$redraw()
  Do $cinst.$objs.pagePane.$objs.empName.$redraw()
```

The buildHolidayList class method builds the list of holiday requests for the selected employee and redraws the form.

Entry Field

Group	Icon	Name	Description
Entry Fields	Ab	Entry Field	Standard edit field for data entry o

The JavaScript **Entry Field** (or Edit control) is a standard Single Line Entry field which you can use to display data or allow the end user to enter data into a Remote Form, such as the Name and Address fields on a Contact form. You can add a text label to each Entry field on the form, to identify its purpose, or you can use the $label property to add a dynamic label. You can add a content tip to each entry field to help the end user fill out the form. The following screenshot shows entry fields with standard label objects and content tips.



Title: Mr
First Name: Bob
Last Name: Jones
Email Address: email address
Telephone: telephone

Figure 167:

Many of the example apps under the **Samples** option in the **Hub** in the **Studio Browser** use entry fields, as well as the apps in the JavaScript Component Gallery, including the **JS Input Border and Button Styles** example app. You can examine these apps and code to see how entry fields can be used.

Dataname: $dataname

The Entry field can handle all types of character or numeric data stored in the *instance variable* specified in **$dataname:** the type of data the edit control can handle will depend on the data type of the instance or column variable assigned to the control. To create an Entry field, you need to drag the Entry field from the Component Store and drop it onto your remote form. You can enter a suitable name in the $name property in the Property Manager, and then you can assign a variable in the $dataname property, that is, it must be an instance variable for a remote form edit control. You can enter the name of an existing variable, or you can enter a new variable name in the $dataname property in the Property Manager, then press Return and the New Variable Name dialog will open allowing you to define the new instance variable; in this case the instance variable is added to the current remote form class (visible in the Method Editor under the Instance tab in the Variables pane).

If you use one of the wizards to create a remote form, the edit controls are added to the form and the required variables are assigned to their $dataname properties automatically.

Single or Multi-line Fields

When the $issingleline property is kTrue (the default), the edit control only allows data entry on a single line, so longer text entries will scroll the entry field to the right. You can create a Multi-line Entry field by setting $issingleline to kFalse, and resizing the height of the field downward to allow data to be entered on multiple lines, using standard line-wrap.

Text Properties and Field Styles

You can set the text style or font of an Entry field using the properties under the Text tab in the Property Manager (when the Advanced option is enabled). The text or font properties are:

Property	Description
$font	The font for the text inside the entry field, either a single font name, or more typically, a list of fonts from which a font is selected depending on the fonts that are installed on the client, e.g. Verdana, Arial, Helvetica, Sans-serif. The fonts will be different for each platform providing a native appearance for each platform
$fontsize	The font size for the entry field, an integer, e.g. 10 to represent 10pt font
$fontstyle	The font style for the entry field: kPlain (the default), kBold, kItalic, kUnderline, kLineThrough, kSemiBold
$textcolor	The text color of the text inside the entry field, either kColorDefault or another color from the current JS Theme, or a specific color specified in the color picker
$align	The alignment or justification of the text inside the entry field: kCenterJst, kLeftJst (the default), kRightJst

See below for content tips and dynamic text label properties.

As an alternative to controlling the font or text of an Entry field using the text properties, you can use a pre-defined Field Style specified in the **$fieldstyle** property. The field styles are set up in the #STYLES system table in the library and are available for many control types, and can contain settings for: $font, $fontsize, $fontstyle, and $textcolor. Setting up field styles for JS Entry fields is the same as for Window class Entry fields which is described here: Field Styles.

Dynamic Labels

You can add a dynamic or “floating” label to Edit fields (and droplists, or the editable part of combo boxes), rather than using separate text labels for the fields in a form. The following properties are available to support dynamic labels:

Property	Description
$label	The label text
$labeliscontenttip	If true, the label is shown as the content tip while the control is not focused and does not have any text content
$labelfontsize	The font size for the minimized label text
$labeltextcolor	The label text color. By default, this is the border color tinted with the text color
$labelhottextcolor	The label text color when the control is focused. By default, this is the same as the focused border color
upto 35989 $labelposition	The position of the label when not shown as the content tip inside the field, a constant: kJSLabelPosBorder (the default), kJSLabelPosAbove, kJSLabelPosLeft
asof 35990 $labelposition	The position of the label when not shown as the content tip inside the field, a constant: kJSLabelPosBorder (the default), kJSLabelPosAbove, kJSLabelPosLeft, kJSLabelPosInside

To enable a dynamic label, you need to add the label text to the **$label** property for the control. Once you have added text to the $label property, you can double-click on the label to edit the label text (pressing Return confirms an update). By default, the text label is inset into the top border of the control ($labelposition = kJSLabelPosBorder), unless $labeliscontenttip is true, but $labelposition can be changed to above or left of the control.

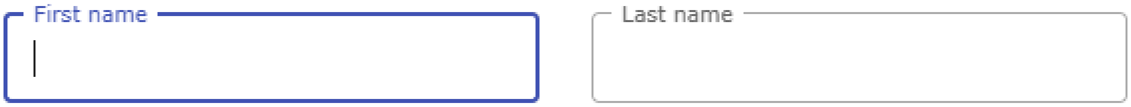


Figure 168:

If $labeliscontenttip is true, and the field does not have the focus or any content, the text in $label is displayed inside the field like a content tip (see Lastname below). In this case, when the focus passes to the field, the label will jump from within the field area to above the field (see Firstname below). You can use this method of adding content tips to fields as an alternative to using the $::contenttip property.

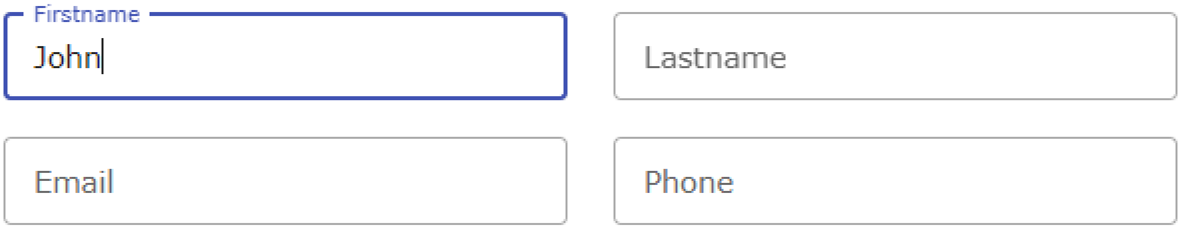


Figure 169:

By default, $inputborderstyle is set to kJSInputBorderStyleOutlined and the label is displayed above the field inset into the border. However, you can set $inputborderstyle to kJSInputBorderStyleUnderline, in which case the field is displayed with underline only (the border is hidden). When the field gets the focus, the label will jump to above the field (see Email field).

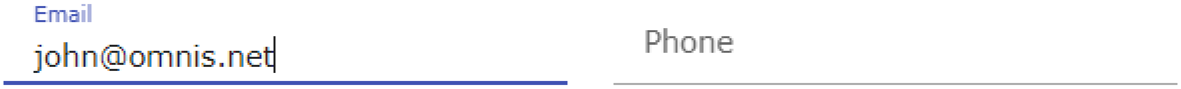


Figure 170:

Content Tips

If you are not using the $label and $labeliscontenttip properties, you can use separate label objects and the $::contenttip property to add descriptive text to entry controls. For example, for a Last name field you could enter ‘Enter your last name’ into $::contenttip to prompt the end user for their last name. As soon as the end user starts to type something into the field the content tip will disappear.

The **$contenttiptextcolor** property is the text color used for the content tip text, or the text in $label when displayed as a content tip, for an edit field, droplist, or the editable part of a combo box. This property applies when using the $::contenttip property or $labeliscontenttip in conjunction with $label.

Password Fields

When kTrue, the $ispassword property ensures a place-holder character is displayed when the end user enters something in the field (only applies when $issingleline is kTrue).

Borders

The $effect property lets you set the type of border for the edit control and its value is one of the kJSborder… constants. The kJSborderDefault setting means the control has the default border type as specified by the current client operating system and browser type. For some clients the border may change when the state of the control changes.

The $borderradius property lets you add rounded corners to the edit control. A single value specifies the radius for all four corners, but you can specify a different value for each corner by specifying four-pixel values separated by – (hyphen), in the order topleft, topright, bottomright, bottomleft. If bottomleft is omitted, the topright value is used. If bottomright is omitted, the topleft value is used. If topright is omitted, the topleft value is used.

Horizontal and Vertical Padding

The property $horzpadding allows you to add extra horizontal padding, in pixels, inside the control; when applied this property adds padding on the left and right of the text within the edit control.

Existing users should note: when a library is converted to Studio 10.2, the $horzpadding property for all JS Entry fields will be set to 4 automatically if they were previously set to 0, which is the default for all new Entry fields; if $horzpadding is set to any other value it is not changed. After conversion, you can change the value of $horzpadding.

The $vertpadding property allows you to add vertical padding above and below the text inside the control's border; the property only applies when $issingleline=kFalse as single line edit controls are vertically centered.

Linked Lists

The $linkedobject property allows you to link a list control to the edit control to create a special type of list called a “Linked List” which updates itself automatically as the user types into the edit control: see the *List Control* section for more information.

Events

The Edit Control reports the **evBefore** and **evAfter** events, so you can detect when the focus is about to enter or leave a field in the $event method and process the event accordingly: note evAfter is only reported *if the data in the control has changed*. You must enable any of the events for the Entry field in its $events property before any of the events are reported.

In addition, the evKeyPress event is reported allowing you, amongst other things, to create a “Linked List”: see the Linked Lists section.

Soft Keypad Input Type

Many iOS and Android devices have different software keypad layouts which are displayed depending on the type of data required, that is, the keypad content and layout adapts to the content required. For example, if numeric content is required, a numeric keypad is displayed. The $inputtype property allows you to specify which keypad is displayed on touch devices depending on what type of content you wish the end user to enter into the edit control. This property only applies to touch devices and in this case $inputtypetouchonly is set to kTrue by default, and it only applies if $ispassword is false (if true the default keypad is shown).

- **$inputtypetouchonly**
  When true (the default for touch devices) the specified $inputtype is applied on touch devices, otherwise it is ignored
- **$inputtype**
  The HTML input type used by the edit field. The browser may give this special handling, e.g. by popping up a specific software keypad. (The property is ignored if $ispassword is enabled.) The input type is specified using a constant:
  kJSInputTypeDefault: the standard Qwerty keypad (in most cases)
  kJSInputTypeNumber: the standard Qwerty keypad flipped to numbers
  kJSInputTypeTelephone: the telephone number keypad
  kJSInputTypeEmail: the Qwerty keypad plus the @ and dot keys
  kJSInputTypeUrl: the Qwerty keypad plus dot, forward-slash, and '.com' keys
  kJSInputTypeDate: shows a date picker; see below

If $negallowed is true, the 'number' type is not selected as this does not guarantee the presence of a minus key. You should note that a touch device's virtual keyboard is more likely to use a numeric keyboard if $negallowed is false.

Input Masks

You can control the format of the data entered into an Entry field by defining an *input mask* using the $inputmask property. If the user enters an invalid character, the control will briefly become highlighted and the input will be rejected. For edit fields of character type, the data variable will contain mask characters. For number/integer fields, the data is the unmasked number value.

The #JSMASKS system table stores the input masks for JS Entry fields for the library, which are also accessible in the $javascriptinputmasks notation group.

There are a number of differences between the existing Masked entry field on the thick client and input masks for JS Entry fields:

- On the thick client, the user must complete the masked entry field before focus can leave the field. This is not the case with JS masked edit fields - fields can be left partially filled.
- JS input masks do not support any of the 'control characters' which can be used on the thick client.
- The JS edit control does not have a $formatstring property (like the thick client masked entry field).
- The JS edit control has two unique properties: $inputmaskguide and $maskvaluevalid.
- JS input masks can be changed dynamically as the user types using the $processmask client method.
- There is visual feedback when entering invalid characters in a masked JS edit field.

Incompatible input types are prevented from being used with input masks. For example, the kJSInputTypeNumber and kJSInputTypeEmail values of $inputtypes are incompatible with input masks. If $inputtype is one of these values, and $inputmask is set, the input element will use the text type (effectively kJSInputTypeDefault).

$inputmask

The value of $inputmask may contain a combination of fixed and special characters. Note that underscores cannot be used as these are used as placeholders.

Special character	Description
#	Any digit
@	Any character
a	Any letter
A	Any uppercase letter
n	Alphanumeric
N	Alphanumeric, uppercase
"ABC"	Any character from list
"A-D"	Any character from A to D inclusive
\	(back slash) Escape character (next character is displayed literally, use to escape special mask characters, double quotes or backslash)

$inputmaskguide

The $inputmaskguide boolean property controls whether or not a guide is shown. If true, placeholder and non-placeholder mask characters are always displayed. If false, placeholder characters are hidden, and mask characters are only shown when the user reaches them as they type. The property is false by default.

$maskvaluevalid

The $maskvaluevalid property is a boolean, read only, runtime only property. A value of kTrue indicates that the field is completed, and therefore valid.

$processmask

A client method named $processmask can optionally be added to an edit control. This allows the mask to be changed as the user types. The method is called any time the value in the field changes, and receives a parameter pInput which contains the user input. Note that the user input parameter could contain anything as the event is sent before mask validation occurs (the mask needs to be updated before it can validate input). As a general use case, $processmask could be used to create the effect of optional characters.

Date Picker

The constant **kJSInputTypeDate** can be assigned to the **$inputtype** property to allow the end user to select a date using the Date Picker. When $inputtype is set to kJSInputTypeDate, and $inputtypetouchonly is set to false, a date/time picker will be used to pick the value for the Entry field. $dataname must be set when using kJSInputTypeDate and other input types such as kJSInputTypeNumber.

The format of the date picker should be calculated from $dateformat ($dateformatcustom if $dateformat == kJSFormatCustom). If $dateformat is kJSFormatNone, then the control attempts to fall back to the dataname subtype.

The **$datepickeroptions** property allows you to customize display options in a calendar style popup date picker, plus the ability to display the isoweek number has been added. $datepickeroptions (and $columndatepickeroptions in the datagrid) is an integer type property with the ability to switch on/off the $showheading, $showmonthnav, $showweeknumber properties in the popup date picker in Entry fields and Data grids. The constants for $datepickeroptions are: kJSDatePickerOptionsShowHeading, kJSDatePickerOptionsShowMonthNav, kJSDatePickerOptionsShowWeekNumber, which can be selected in the Property Manager or added together to set the options in your code. The kJSDatePickerOptionsShowWeekNumber option shows the isoweek number down the left-hand side of the calendar layout.

By default, kJSDatePickerOptionsShowHeading and kJSDatePickerOptionsShowMonthNav are set to true and kJSDatePickerOptionsShowWeekNumber is set to false to maintain behavior in previous versions.

Custom Date Formats

You can specify multiple date formats in the $dateformatcustom property for entry fields in a remote form, which allows end users to input a date using one of a number of possible formats, rather than being limited to a single date format. The multiple date formats can be entered into $dateformatcustom separated using "|" (the pipe character), for example:

```
D/M/y|D m y|d-m-y|D/M/Y|D/m/y|D-M-y|D M y
```

When parsing data entered by the user, the client uses each format in order, until one successfully matches the user input. The client uses the first format in the list to format the data for display.

Numeric Data Entry Validation

Entry fields (and data grids) validate numeric data as it is entered, for JavaScript remote form variables with a numeric data type. When the end user tries to enter invalid data into the field, such as an alphabetic character, the data is rejected, i.e. it cannot be entered into the field, and the field is highlighted momentarily to indicate an error (the default action is to show a red border).

When leaving the entry field, the value is normalized: so for integer data it is constrained to the valid range or for other numbers it is rounded to the correct number of decimal places; also, leading zeroes are removed, and so on.

Negative Numbers and Uppercase

The $negallowed property for Entry fields allows for the display of negative number values. If this is kFalse and the end user tries to enter a negative sign, the sign will be rejected.

When set to kTrue, the $uppercase property forces all character data to be shown in uppercase.

Zero Values

When the $zeroempty property is set to kFalse (the default), a zero character is shown in a numeric type entry field. Otherwise, you can set $zeroempty to kTrue to ensure a numeric edit field is empty when the variable value is zero. You can have greater control over the display of numbers using Input masks.

Multiline Edit Scrolling

Text wrapping for the Multiline Edit field is prevented if the $horzscroll property is enabled (kTrue). However, if $autoscroll is true, then text wrapping does occur (since $autoscroll is on by default).

Shortcut Keys

There are a number of shortcut keys defined in the Entry field that allow end users to select text and move the insertion point within the control (the shortcuts also apply to Window class Entry fields).

The shortcut keys are stored in the Omnis preference **$keys** which can be edited in the Property Manager (click on the Prefs option in the top level of the Studio Browser). The shortcut keys for Entry fields are stored in a configuration file called 'keys.json' and located in the Studio folder (this is the same file containing the shortcut keys for the Method Editor). The file is created the first time you edit the shortcuts in the Property Manager and click OK.

Shortcut key	Action
Alt+End	End of Text Alternative
Alt+Home	Start of Text Alternative
Ctrl+Alt+DownArrow	Scroll Down
Ctrl+Alt+LeftArrow	Scroll Left
Ctrl+Alt+RightArrow	Scroll Right
Ctrl+Alt+UpArrow	Scroll Up
Ctrl+DownArrow	Paragraph Down
Ctrl+End	End of Text
Ctrl+Home	Start of Text
Ctrl+LeftArrow	Backwards Word
Ctrl+RightArrow	Forwards Word
Ctrl+UpArrow	Paragraph Up
End	End of Line
Home	Start of Line
PageDown	Page Down
PageUp	Page Up

Auto Correction, Capitalization, Completion

You can control the Automatic Correction, Capitalization And Completion of text entered by the end user into an edit control. This functionality is built into the browser whereby text is 'corrected' or updated automatically: note that not all browsers support all of these functions, so you should check support in individual browsers on different platforms.

You should note that Auto correction, Auto capitalization and Auto completion (the properties $autocorrect, $autocapitalize & $autocomplete) only work in Omnis when they are enabled on the client Mac computer. Note also that $autocapitalize only applies when using a virtual keyboard on a device.

Auto Correction

If true, the $autocorrect property specifies that text entered into a JavaScript Entry field is auto-corrected, *as the end user types into the field*. Currently this feature is only implemented in Safari and iOS browsers (note some browsers, such as Chrome, will highlight incorrectly spelled words, but not correct them automatically). This property defaults to kTrue for backwards compatibility.

Auto Capitalization

The $autocapitalize property controls whether or not text typed into an edit field is capitalized, as appropriate, automatically. Possible values include:

- **kJSAutoCapitalizeSentences**
  Automatically capitalize the first character of new sentences (default and previous behaviour)
- **kJSAutoCapitalizeWords**
  Automatically capitalize the first character of each word

- **kJSAutoCapitalizeNone**
  No automatic capitalization

Auto Completion

The $autocomplete property controls whether or not text is completed automatically. If true, the browser will attempt to complete text based on content previously entered into this type of field (e.g. name type fields will display a list of names based on the first letter typed): the browser will display a list of suggested text for the end user to select from. There may be many possible values for some types of field, which will be based on values previously entered into any website for a field with the same ‘autocomplete’ value, e.g. ‘email’.

Content Selection

The **$setselection** method allows you to select a range of characters within the Entry field; note the method only works on the client.

- **$setselection**(iFirstSel[,iLastsel]))
  Sets the focus on and selection range of the content in an Entry field. If iLastSel is omitted selection will occur to the end of the edit field. Returns the selected text.

$setselection has two parameters, both Integer, iFirstSel and iLastSel to set position of the characters to be selected within the edit control. iLastSel selects up to, *but not including,* the character specified, and if omitted the content in the edit field is selected to the end. The method returns the content selected.

You can also use $cinst.$setcurfield(vNameOrIdentOrItemref [,bSelect=kFalse]) to put the focus in the specified field, and if bSelect is kTrue and the client supports it, the contents is selected. This can be used on the client or in a server method; see Remote Form Methods for more info.

Dictation Text Entry

You can enter text into an edit field using the built-in Dictation feature on macOS, which tries to convert audible speech into meaningful text. To allow dictation to occur the focus must be in the edit field, which must itself be editable, i.e. not disabled, and dictation must be enabled on the client computer. Dictation is available in Single- and Multi-line edit fields, the edit part of Combo boxes, and edit fields in Complex grids in remote forms (and window classes), that is, wherever text input is required.

Enabling Dictation

Support for Dictation is turned on in Omnis by default, but you can change it in the config.json file. There is a “useDictation” item in a “macOS” section of the config.json file, which is set to true to enable dictation; note you have to quit Omnis to change the config.json file, and any change will be effective when you restart Omnis.

```
"macOS": {
  "useDictation": true
  }
```

Using Dictation in Edit fields

To enter dictation mode, place the cursor in the edit field and select the *Start Dictation* option from the Edit menu on macOS, or press the Function (Fn) key twice. This will open the dictation popup (usually in the center of the screen, depending on space) and put the computer in listening mode. Dictation can be stopped or cancelled by clicking on Done in the popup, or using the *Stop Dictation* menu option.

Dictation Levels

There are two levels of dictation provided by macOS: *Standard* or *Enhanced*. These can be enabled from System Preferences->Keyboard->Dictation, or on older systems System Preferences->Dictation & Speech.

Standard dictation (the default) requires an internet connection and provides speech to text translation using Apple’s servers. On older systems, the text is not translated until the Done button is pressed on the popup. On newer systems text is translated and placed into the field while the end user is speaking. Dictation will end automatically when text is entered from the keyboard or the field loses the focus.

Enhanced dictation requires the enhanced dictation engine to be downloaded, which is approximately 500MB for each language pack. This will then provide local machine based translation. Features of enhanced dictation are live feedback and offline support. With live feedback the text is rewritten while speaking. Enhanced dictation also provides spoken dictation commands such as “Select All”, “Cut that”, “Move left”, and so on. When enhanced dictation has been started it is possible to change the currently focused edit field and move the popup to the new field and continue to dictate. It is also possible to type and dictate at the same time.

Tooltips and Carriage Return

Text in tooltips will wrap if it contains a carriage return character or other wrapping characters when the text width of the tip would exceed a third of the screen width. In previous versions, tooltips only used CR to line wrap when the width of the tip was greater than half the screen width.

In addition, the CR character is no longer displayed. However, any other control characters (characters less than space, or the character 0x7f) are displayed using the Unicode control character page.

This change also applies to tooltips for window class controls.

File Control

Group	Icon	Name	Description
Media	⬆	File Control	Allows end users to upload or dow files

The **File Control** allows end users to upload or download files inside a remote form: from Studio 10 the control allows multiple files to be uploaded. The control itself is invisible and to enable the upload or download functions, you need to assign an action at runtime, to the $action property. The Upload and Download actions are represented by the constants kJSFileActionUpload and kJSFileActionDownload.

There is an example app called **JS File Upload/Download** in the **Samples** section in the **Hub** in the Studio Browser showing how you can upload and download files, and the same app is available in the JavaScript Component Gallery. In addition, the **Contacts** example app, under the **Applets** option in the Hub, uses the file control to upload a contact photo.

When in **Upload** mode the File control opens a standard Upload dialog to allow the end user to select a local file or files to upload. There are various properties to allow you to change the text and error messages on the Upload dialog, including $choosefilesbuttontext that allows you to specify the text on the ‘Choose Files’ button. In addition, you can setup the File control to receive files that are dropped onto the File control in the remote form itself.

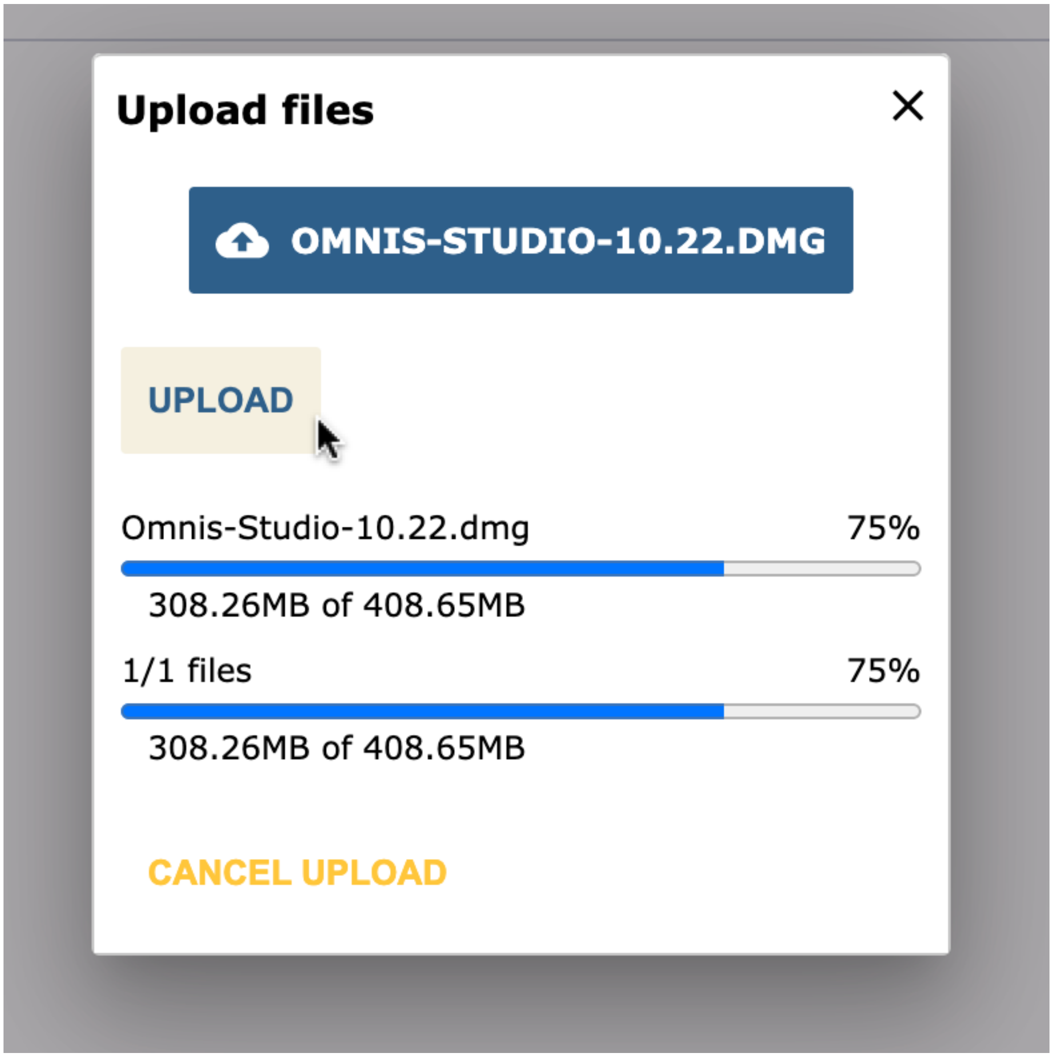
When in **Download** mode, the File control can provide a standard hyperlink pointing to the file to be downloaded, or you can assign a row variable to the $dataname of the control containing the binary data of the file to download, or a list for multiple file downloads. The download function is supported for mobile devices, and if the browser can interpret the contents of the file it is shown in a new browser window or tab.

The $maincolor property is the main color used throughout the control (upload area, upload button, progress bars, completion indicators, upload spinner).

Mobile limitations or issues

Support for the Upload or Download function on mobile devices depends very much on the device itself and the mobile operating system. Therefore, if you intend to include an upload or download function in your app, you should test your app thoroughly on all devices you wish to support. With this in mind, we are aware of the following limitations or issues regarding different mobile operating systems.

Upload does not work on iOS because the input element to select a file does not become enabled since iOS does not support it. Download works via the hyperlink because mobile Safari has been implemented to support hyperlinks. However, the other download mechanism does not work: on iOS for example, the download actually transfers the data to the client, but then the browser does not carry out any action with the data, so the downloaded file is lost.



Upload files

Omnis-Studio-10.22.dmg 75%

308.26MB of 408.65MB

1/1 files 75%

308.26MB of 408.65MB

Figure 171:

Uploading Files

To enable the upload function, you need to set up a list to receive the uploaded files and you must set the $action property to kJSFileActionUpload. In addition to using the File control to allow the end user to select files by clicking an Upload button, you can switch the control to receive files that are dropped onto the control; see Dropping Files.

Setting up the file list

The $dataname property of the File component needs to be set to a two-column row variable: Column1 should be of type List, and column2 should be of type Character ($dataname should be set before the upload action is triggered). The value assigned to column2 should be the *name* of a task variable of Binary type which will receive the binary data when the upload is complete. Column1 will receive MIME header list information when the upload is complete.

Setting the Upload action

To trigger the upload dialog, you need to assign kJSFileActionUpload to the $action property. This action opens a file upload dialog which has two formats: one for recent web browsers, and the other for browsers which do not support XMLHttpRequest2, i.e. Internet Explorer and Opera. When a file has uploaded an **evFileUploaded** event is generated and the List and Binary task variable assigned to $dataname are populated. When the dialog closes the control generates evFileUploadDialogClosed.

The $maxfileuploadsize specifies the maximum size in bytes of a file to be uploaded to the server. Zero means no limit. Some clients (IE and Opera) cannot enforce this limit. See example below for details on how to upload a file.

While the Upload dialog is open, you can close it by assigning kJSFileActionCloseUpload to the $action property (note this does not generate evFileUploadDialogClosed).

On the upload dialog the file name is displayed above the progress bar, to the left, the percentage is shown above to the right, and file upload size information shown below as before. File sizes displayed to the user are in shown in the appropriate unit so when 1000 bytes is exceeded it changes to kB, 1000 kB changes to MB and 1000 MB changes to GB.

The $clearsfileselection property clears the last uploaded file. When set to true, the current file selection is cleared automatically after an upload has completed.

Upload File Type

The $uploadtypes property allows you to filter the file types that can be uploaded. The property accepts a comma-separated list of file extensions or MIME types, for example, the string '.png, .jpg, .jpeg' would allow PNG or JPG files, or 'image/*' to allow any image files.

Uploading Multiple Files

The File control allows multiple files to be selected for uploading by setting $allowmultiple to kTrue. It is not supported by some browsers, just like $maxfileuploadsize (IE9 and below, Opera).

The properties $maxbatchuploadsize and $maxbatchuploadsizeerrortext work similarly to $maxfileuploadsize to impose a limit of the total amount of data to be uploaded. This works independently of $maxfileuploadsize so you can impose a limit on single or multiple file uploads. In addition, $uploadedprogresstextbatch has been added to show the progress of the batch of files, and works similarly to $uploadprogresstext.

Error messages shown when file sizes are exceeded, for single or batches of files, give the user feedback on what the size limits are and lists the offending files exceeding the limit.

Multiple file upload dialogs display the same information for each single file, while another progress bar shows progress through the batch of files, including how many files have uploaded out of the total.

Using Timer and File controls

Note that if a Timer control is present on the page, timer events will not occur while a file upload dialog is open, or while file download is in progress.

Localization

All the text and labels in the File control can be translated via the jOmnisStrings object in the JavaScript client. See the main Localization section for more info.

Upload UI

The **$choosefilesbuttontextpos** property allows you to position the text label in the Upload UI. It can be assigned a kJSFileUploadLabelPos... constant to specify whether the label is shown at the Top, Right, Bottom, or Left of the icon, or it can be set to None to hide the label (the constants are kJSFileUploadLabelPosTop, kJSFileUploadLabelPosRight, kJSFileUploadLabelPosBottom, kJSFileUploadLabelPosLeft and kJSFileUploadLabelPosNone).

The **$choosefilesiconid** property allows you to specify an icon in the Upload UI; the default is the file_upload icon from the material iconset set to 48x48 size. If a themed SVG is used, it will take on the same color as $maincolor. Note that if this icon is cleared, there will be no indeterminate spinner shown while uploading files.

Dropping Files

You can setup the File control to receive files that are dropped onto the **File control** to upload them. In this case, the input area is larger, and a generic progress spinner is displayed, which replaces the choose files icon while uploading. This may be useful if you wish to opt for a simpler interface by switching off the progress details (see $hideuploadprogress and $hideuploadprogressbatch below). In addition, you can use the file control as an upload area inline on the form, and files upload automatically instead of the end user having to click another button once the files have been chosen.

To allow files to be dropped the **$showinline** property must be set to true. If true, $hyperlinktext and $hyperlinkurl will be ignored, and kJSFileActionUpload assigned to $action (used to open the file dialog) will also be ignored. If **$autoupload** is true, no Upload button will be shown, and instead files will be uploaded as soon as they are chosen.

If **$hideuploadprogress** is true, the upload progress area is hidden. Plus **$hideuploadprogressbatch** allows you to hide the total batch upload progress area. The spinner is displayed if both upload progress areas are switched off.

The **$uploadprogresstextcolor** property is the color for the text in the progress elements.

Downloading Files

URL downloads

In its simplest form, the File control can provide a hyperlink to allow the end user to download a file located on the internet. In this case, you would set the $visible property of the control set to kTrue, set $hyperlinkurl to the URL of the file, and $hyperlinktext to the text for the hyperlink. The URL does not have to be a file download URL, it can be any file on the internet and will open in a new browser window or tab. The File control will display the download as a standard hyperlink on the remote form.

Single File downloads

You can also use the File control to download a file from a binary file or one stored in your database. To enable this functionality, you need to assign kJSFileActionDownload to the $action property. To download a single file, the $dataname property of the File control is set to a row variable, with Column1 as the file name, Column2 the media type (e.g. text/plain;charset=utf8), Column3 the name of a remote task variable which should be a binary containing the file data or character containing a file path, and Column4 is an ‘Identifier’ column (Character type) containing a name for the file.

Multiple File downloads

You can specify a list in $dataname, instead of a row, to provide a list of files to be downloaded. The definition of this list is identical to the row, with each line in the list representing each file to be downloaded. The optional fourth ‘Identifier’ column can contain a unique name for each file.

You can use the **evDownloadSent** event in the $event method for the File control to loop through the file list to download each file. This method is in the JS File example in the Samples section under the Hub.

```
# iFiles is defined in $construct with cols iFileName, iFileLen, iFileBinary
On evDownloadSent
  Do iFiles.$next(iFileLineRef,kTrue,kFalse) Returns iFileLineRef
  If iFileLineRef
    Do method downloadFile
  End If
```

The downloadFile method is:

```
Do iFiles.$loadcols()
Calculate tJSFileBinData as iFileBinary
Do iJSFileRow.$define(iJSFileName,iJSMediaType,iJSVariableName)
Do iJSFileRow.$assigncols(iFileName,'application/octet-stream','tJSFileBinData')
Do $cinst.$objs.JSFormFile.$action.$assign(kJSFileActionDownload) Returns #F
```

evDownloadSent Event

The **evDownloadSent** event indicates if the download was successful and identifies the file that was downloaded. The event receives two parameters:

- **pSuccess**
  Whether or not the download was successsfully sent.
- **pIdentifier**
  The value of the Identifier column (the fourth column, if provided) in the download list/row which was assigned to $dataname to initiate the download which has now been sent.

The evDownloadSent event will be triggered when the server has sent all of the data for a file to the client. Note that Omnis sends the data in a single chunk, so the client may still be processing the data at this point. However, at the point that the event is fired, the server has sent all of the data, and the task variable containing the binary data (or file path) can now be re-used.

Example

A File control is used in the jsContacts form in the **Contacts** sample app to allow end users to upload a photo of their contacts. The File control is placed somewhere on the form, its $dataname is set to iFileRow, an instance row variable, and the **evFileUploaded** and **evFileUploadDialogClosed** events are enabled in the $events property of the control. The iFileRow variable is setup in the $construct method of the form, as follows:

```
Do iFileRow.$define(MimeList,"Binary Variable")
Calculate iFileRow.C2 as "tData"
```

MimeList is a local list variable, and tData is a task variable of type Binary – it’s important to note that the second column of iFileRow is of type Character, and is set to the ***Name*** of the binary task variable. Elsewhere on the form is a picture control to display the photo, its $dataname is iPicture, and a button to allow the end user to select an image file and initiate the upload process. The code for the button is:

```
On evClick
  Do $cinst.$objs.fileControl.$action.$assign(kJSFileActionUpload)
```

This assigns the kJSFileActionUpload action to the fileControl object which opens the Browse/Upload dialog. The File control has an $event method which detects when a file has been uploaded and the upload dialog has been closed:

```
On evFileUploaded
  Calculate iPicture as tData
# transfer the pic data to iPicture
On evFileUploadDialogClosed
  Do method setPicBtnTitle (kHavePic) ## changes the button text
```

When an image is selected by the end user and the file is uploaded the image data is loaded into the tData task variable, as defined in the second column of the iFileRow row variable, which is then transferred to the iPicture variable assigned to the picture field on the form. The Save button saves the contact record including the image file uploaded by the user.

Floating Action Button

Group	Icon	Name	Description
Buttons	⊕	Floating Action Button	A round button that pops up a list actions when tapped or hovered o

The **Floating Action Button (FAB)** component features a round button that pops up a list of actions when tapped or hovered over, with the first option being a default action. For example, in a form displaying a list of contacts, you could use a FAB to provide options to add a contact (the default action), with further options to edit, call, or email a contact.

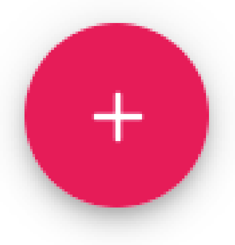


Figure 172:

The FAB is displayed as a circular button containing a '+' icon prompting the end user to tap it or hover over it; the default icon can be replaced by setting $iconid of the button control. In its expanded state, the actions in the list appear to "float" on top of the other content in the form.

Defining the data list

To create an expanded list of actions, the $dataname of the FAB can be assigned a list instance variable with the following columns:

Column	Type	Description
Icon	Character	The URL of the image, generated by calling iconurl(iconid); iconid is the name of an SVG image in an icon set, such as an icon in the material icon set
Action ID	Integer	This should be a unique integer. This will be the value of pActionId in the evClick event, e.g. IDs could be 1, 2, 3, etc.
Label	Character	The label text (this is used as the accessible name of the action if labels are hidden)

There is an example application called **JS Floating Action Button** in the **Samples** section of the **Hub** in the Studio Browser which displays a FAB in the bottom right corner of a remote form. Each line represents an action with the first line, in this case Create, representing the main button in its expanded state.

In the FAB example, a list instance variable iList is assigned to $dataname of the control, which has 3 columns: Icon, ActionID, and Label. The following code is added to $construct of the form, which creates the options shown in the expanded button:

```
Do iList.$define(lIcon,lActionId,lLabel)
Do iList.$add(iconurl("create"),1,"Create")
Do iList.$add(iconurl("save"),2,"Save")
Do iList.$add(iconurl("content_copy"),3,"Copy")
Do iList.$add(iconurl("print"),4,"Print")
Do iList.$add(iconurl("download"),5,"Download")
```

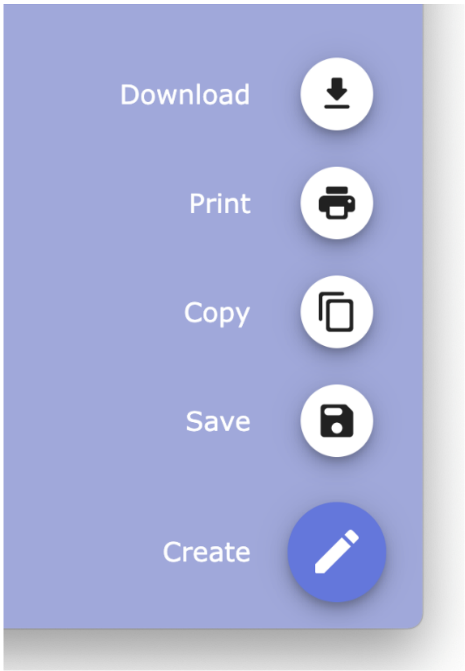


Figure 173:

Properties

The Floating Action Button has the following properties.

Property	Description
$dataname	The name of the list instance variable that defines the expanded actions
$iconid	The icon on the main button in its non-expanded state which replaces the default plus icon; no icon is shown when $iconid is empty
$text	The optional text on the main button. A FAB with text will use the full control area. Without text, it will be circular
$textcolor	The color of text and SVG icon on the main button in its default (non-expanded) state
$textbeforeicon	If true, and the control has both text and an icon, the text is drawn before the icon
$opendirection	The direction in which the expanded actions open, a constant: **kFabDirectionUp** (the default) or **kFabDirectionDown**
$expandedappearance	The appearance of the FAB in its expanded state, a constant: **kFabAppearanceIconOnly** icons only, no labels **kFabAppearanceLabels** (the default) displays the labels for all actions in the list **kFabAppearanceHoveredLabels** each label is displayed when the action is hovered or focused with the keyboard
$labelside	The side on which action labels are displayed, a constant: **kFabLabelSideLeft** (the default) or **kFabLabelSideRight**
$expandedlabelbackground	If true, expanded action labels have a background (default is false)

Property	Description
$expandedmainbackcolor	The background color of the main button when the FAB is expanded. kColorDefault means use $backcolor
$expandedmaintextcolor	The color of text and SVG icon on the main button when the FAB is expanded
$actionbackcolor	The background color of expanded actions
$actioniconcolor	The color of SVG icons on expanded actions
$modalbackcolor	The color and alpha of the modal background when the FAB is expanded; the color picker includes a slider to set the alpha value (0-255), or you can use rgba() at runtime*
$labelbackcolor	The label background color used if $expandedlabelbackground is kTrue
$labeltextcolor	The text color of action labels

*In order to allow $modalbackcolor to be set on the client, the rgba() function can now be executed on the client, which allows you to set the color and alpha value of the property.

The following example FAB has the following properties set (and uses the ice JS theme):
$expandedappearance = kFabAppearanceHoveredLabels
$expandedlabelbackground = kTrue
$labelside = kFabLabelSideRight
$opendirection = kFabDirectionDown

Events

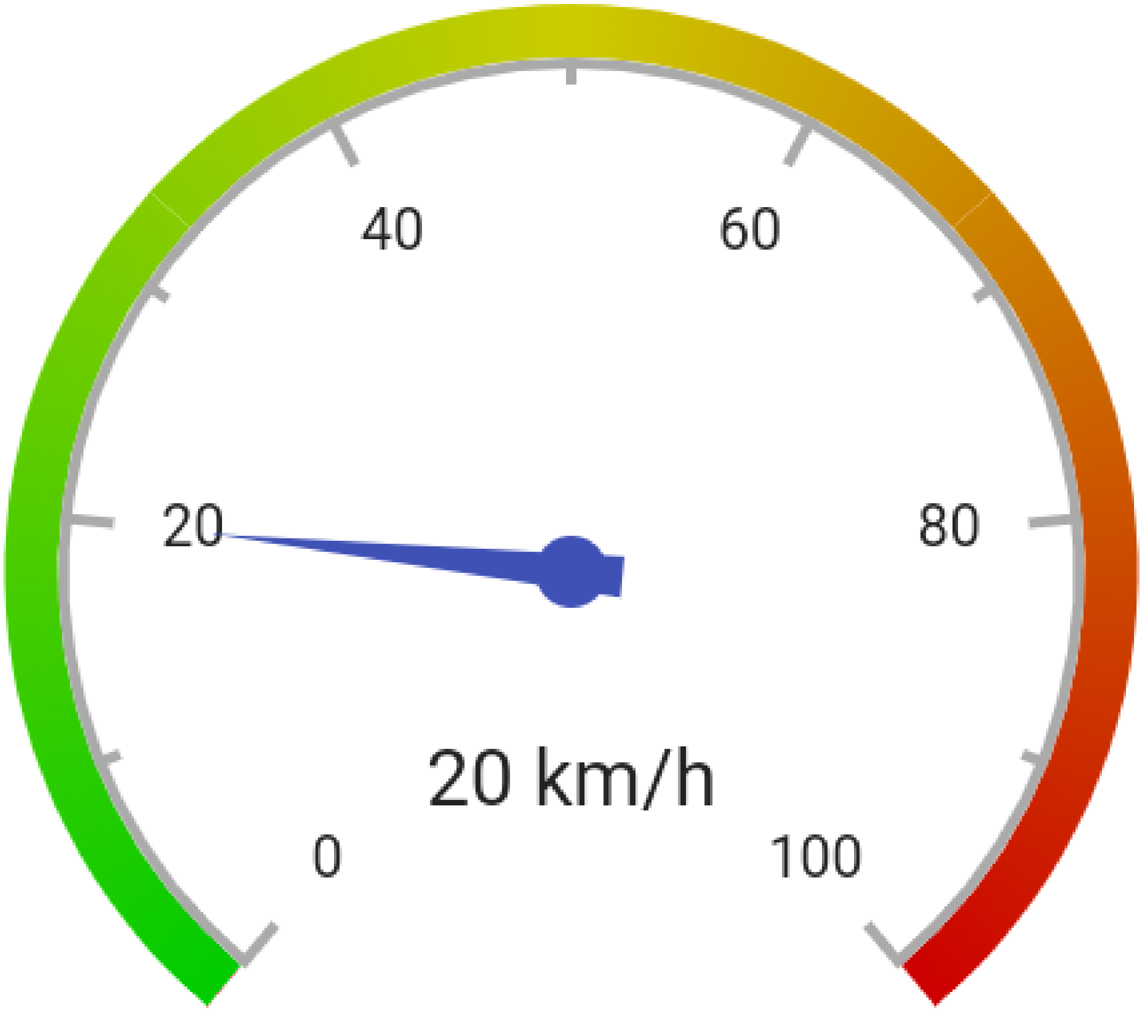
The Floating Action Button reports the **evClick** event, sent when the main button or an expanded action icon is clicked. The **pActionId** parameter contains the value of the clicked action as defined in the second column of the data list. If the main button was clicked in its default state, the value of pActionId is null.

Gauge Control

Group	Icon	Name	Description
Visualization		Gauge	Displays numerical values on a circ linear scale

The **Gauge** component provides a way to display numerical values on a circular or linear scale, with options to customize the appearance and behavior. The Gauge control type can be **Circular, Horizontal** or **Vertical.**

There is an example application called **JS Gauge** in the **Samples** section of the **Hub** in the Studio Browser demonstrating the types of gauge available, including the *Circular* and *Vertical* gauge types, as shown:



A gauge consists of:

- A *circular* or *linear* scale with tick marks and labels that can be customized
- A *needle* or *marker* style pointer to indicate the current value
- A *range* or multiple *ranges* with customizable colors, widths and start/end points; the range is a colored band inside or outside (above or below) of the scale
- A *display value* showing the current value in a formattable string, so you can display units, for example

The current value shown on the gauge is stored in the **$dataname,** which is shown if $alwaysshowdisplayvalue is true, otherwise if false, *the value is only shown* when the end user hovers their pointer over the needle or marker, or the needle or marker is dragged to change its value. You can format the value by setting $displayvalue using a 'sprintf' formatted string. If the $clicktosetvalue property is true, the end user can change the value by clicking on the gauge, otherwise if false, the value can only be changed by dragging the pointer.

You can set the Scale and Range for the gauge control, including the start and end values (e.g. 0 to 100), the *position* of the start and end values (i.e. the angle in a circular gauge), as well as the colors and settings for the tick marks on the scale using various properties; see below for more details about the customizing the Scale and Range.

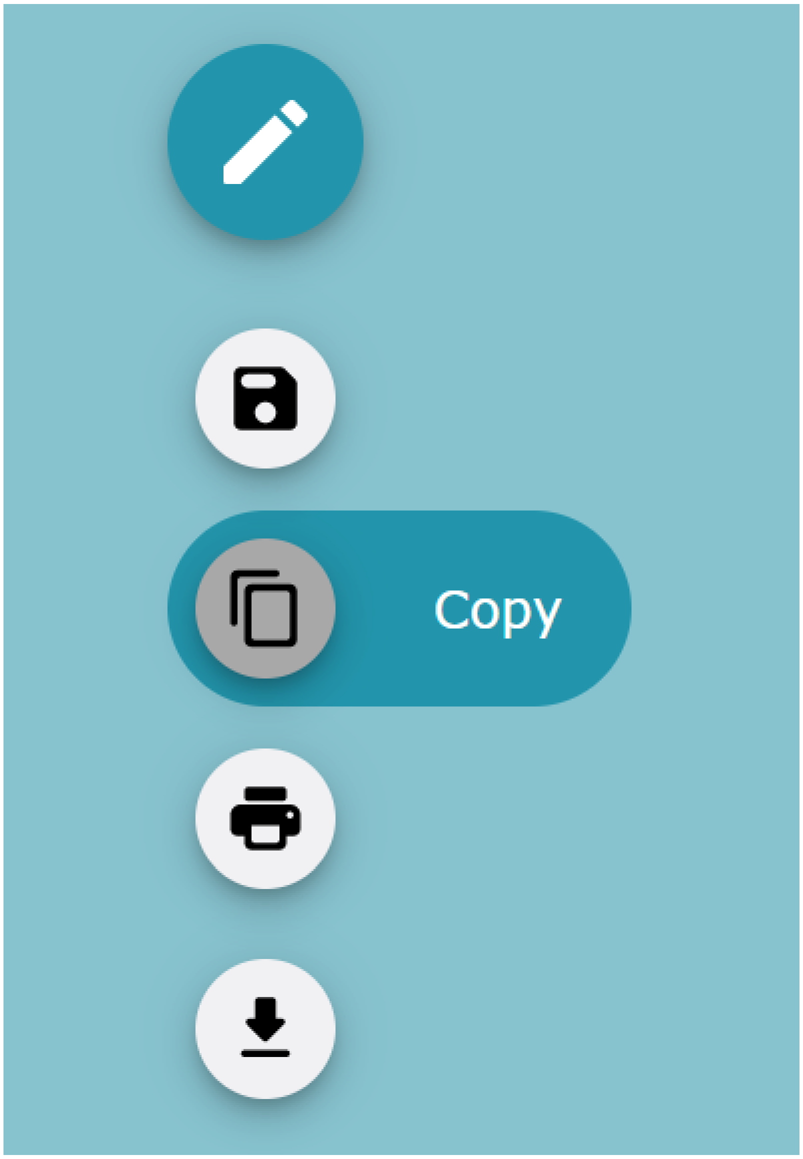


Figure 174:

Properties

The following properties are available for the Gauge control (the range properties are shown after this table).

Property	Description
$dataname	The name of the instance variable that holds the current value. Must be of Number or Integer type
$gaugetype	The type of gauge (Circular, Horizontal or Vertical), a kJSGaugeType… constant: kJSGaugeTypeCircular kJSGaugeTypeHorizontal kJSGaugeTypeVertical
$scalevaluestart	The start value of the scale
$scalevalueend	The end value of the scale
$scaleanglestart	The angle of the start of the scale in degrees where zero is the top. Only applies when $gaugetype is kJSGaugeTypeCircular
$scaleangleend	The angle of the end of the scale in degrees where zero is the top. Only applies when $gaugetype is kJSGaugeTypeCircular
$tickintervalmajor	The interval between major tick lines. Zero means the interval is calculated automatically
$tickintervalminor	The interval between minor tick lines. Zero means the interval is half of the major tick interval
$ticklineheight	The height of major tick lines in pixels, which must be set to make the tick lines visible. Minor tick lines are half of this height
$minstep	The minimum step size on the scale, e.g. set to 5 to allow value to step in multiples of 5. If this is zero or greater than or equal to the scale's range, the major tick interval is used as the minimum step
$clicktosetvalue	If true, the user can change the value by clicking on the gauge. If false, the value can only be changed by dragging the pointer
$reversedirection	If true, the positive direction is reversed
$displayvalue	A formatted string used to display the current value. sprintf syntax with a single format tag for the number, e.g. f km/h; use f and d for a floating and integer number respectively, or F and D in upper case to insert a thousand separator
$alwaysshowdisplayvalue	If true, the current value is always displayed. If false, it is only shown when the pointer is hovered or dragged
$circularpointertype	The style of pointer used when $gaugetype is kJSGaugeTypeCircular, a kJSGaugePointerType… constant: kJSGaugePointerTypeDefault kJSGaugePointerTypeNeedle kJSGaugePointerTypeMarker
$opposeaxis	If true, the position of the axis is opposed, e.g. scale on circular gauge is shown on the outside

Property	Description
$opposeranges	If true, the position of the ranges is opposed
$padding	The padding from the scale line to the edge in pixels. 1 to 4 pixel values separated by -. Possible values: [all sides], [vertical]-[horizontal], [top]-[horizontal]-[bottom], [top]-[right]-[bottom]-[left]
$markeroffset	The offset in pixels of the marker-type pointer from its default position
$rangeoffset	The offset in pixels of the range from the scale line
$animatechanges	If true, the pointer and display value will animate when the value changes
$hidescaleline	If true, the scale line is hidden
$hideticklines	If true, the tick lines are hidden
$hidescalelabels	If true, the scale labels are hidden
$scalelabelfontsize	The font size for the scale labels
$blendrangecolors	If true, the range colors are blended together, to create a color gradient
$pointercolor	The color of the pointer
$scalecolor	The color of the scale and tick lines
$scalelabelcolor	The color of the scale labels
$hidescaleline	If true, the scale line is hidden
$hidescalelabels	If true, the scale labels are hidden

Events

The **evValueChange** event is triggered when the value is changed by the user clicking on the gauge area or dragging the pointer (needle or marker). The **pNewValue** parameter holds the new value.

Customizing the Scale and Range

The properties under the Range tab in the Property Manager control the range values and appearance.

Property	Description
$currentrange	The current range; set this to access properties for each range section
$rangecount	The number of range sections
$rangecolor	The color of the current range
$rangevalueend	The end value of the current range
$rangevaluestart	The start value of the current range
$rangewidthend	The width of the end of the current range
$rangewidthstart	The width of the start of the current range

To show how you can customize the scale and range for a gauge control, consider the following example that displays temperature values in the range 0 to 100.

The following properties have been set:

- On the General tab in the Property Manger, **$scalevaluestart** and **$scalevalueend** are set to 1 and 100, respectively.
- On the Appearance tab, **$gaugetype** is set to kJSGaugeTypeHorizontal, **$displayvalue** is set to %dC, **$padding** is set to 100-10 (100 at the top to display the customized range, 10 for each side), **$rangeoffset** is -2 (which provides a gap between the range and scale baseline), and **$tickintervalmajor** is set to 20.

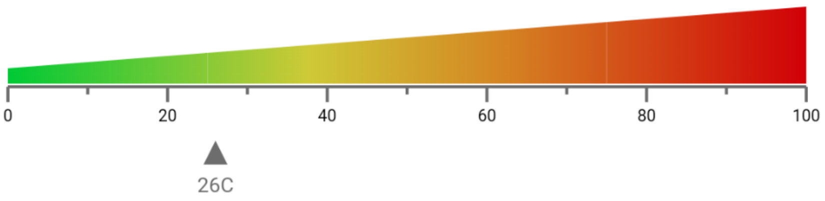


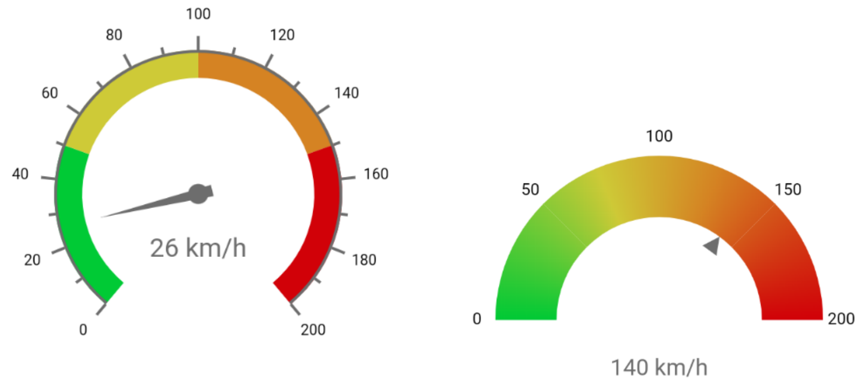
Figure 175:

You can customize the range on the *Range* tab in the Property Manager. The range is not shown by default, so to show a simple range you can set **$rangecount** to 1 and **$rangevalueend** to the same value as $scalevalueend, e.g. 100. However, to specify different colors and widths on the range, like the above example, you need to set **$rangecount** to 4 and specify each range in turn by setting **$currentrange** (a design property) from 1 to 4. The following property values are set for each range section:

$currentrange	1	2	3	4
$rangecolor (r,g,b)	Green (0,202,53)	Yellow (206,202,55)	Orange (213,131,35)	Red (2
$rangevaluestart	0	25	50	75
$rangevalueend	25	50	75	100
$rangewidthstart	10	20	30	40
$rangewidthend	20	30	40	50

In the example, the distinct colors for the ranges are blended automatically by setting **$blendrangecolors** to kTrue, providing a smooth gradation of colors.

You can experiment with the display properties to achieve the gauge appearance you want, including flipping the scale or range using the **$opposeaxis** and **$opposeranges** properties. For example, the circular gauge, shown below left, has $opposeaxis set to kTrue to display the scale and labels on the outside. For the gauge shown on the right, its scale and tick lines are hidden, the pointer type is set to marker (a small arrow), $markeroffset is set to the same value as the range width, while $scalevaluestart and $scalevalueend values are 270 and 90, respectively.



HTML Object

Group	Icon	Name	Description
Media	🌐	HTML Object	Object to display HTML content

The **HTML Object** lets you display an HTML page or fragment in a remote form, such as an HTML link or formatted paragraph. The $html property contains the HTML content for the control. There is an example app called **JS HTML** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

The HTML content for the control must start with an element declaration such as <div…> or <style>. The $wraptext property is set to true by default meaning that the content in the control will wrap (set it to kFalse to stop wrapping): this property sets white-space for the control to 'normal' if true, and 'no-wrap' if false.

HTML Preview

The HTML control has a property, $showruntimepreview, which defaults to kTrue, which ensures the HTML is rendered in the remote form rather than showing the HTML code text. If $showruntimepreview is false, the HTML code text is shown, but it cannot be scrolled inside the control in the design window.

Custom JavaScript Controls (Deprecated)

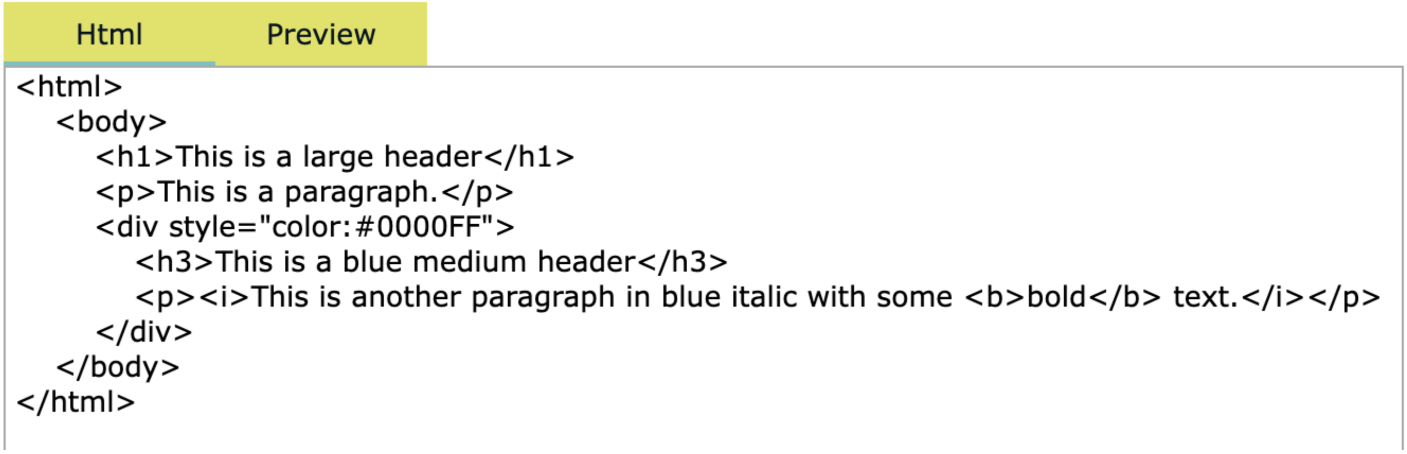
Note: This technique using $ctrlname to add custom controls to a remote form is deprecated. You should instead use JSON-defined controls: see the JSON Components chapter.

You can use a custom JavaScript control on a remote form by embedding it in an HTML Object. To do this, the JavaScript file defining the custom control must be placed in the html/scripts folder and referenced in the HTML file containing your remote form (this can be added to the jsctempl.htm file or the specific HTML page containing your remote form). The name of the custom control class (as defined in its JavaScript file) must then be added to the $ctrlname property of the HTML Object.

If you want your custom control to inherit the HTML control's specific functionality (such as the use of $html and its placeholders feature), your custom control must inherit from ctrl_html, rather than ctrl_base (and do not override this.superclass, instead use a different variable to reference the ctrl_html immediate parent class).

HTML Examples

The example library (in the **Samples** section in the **Hub** in the Studio Browser) shows how you can use HTML to format text using tages and styles, and embed the content into a Remote form.



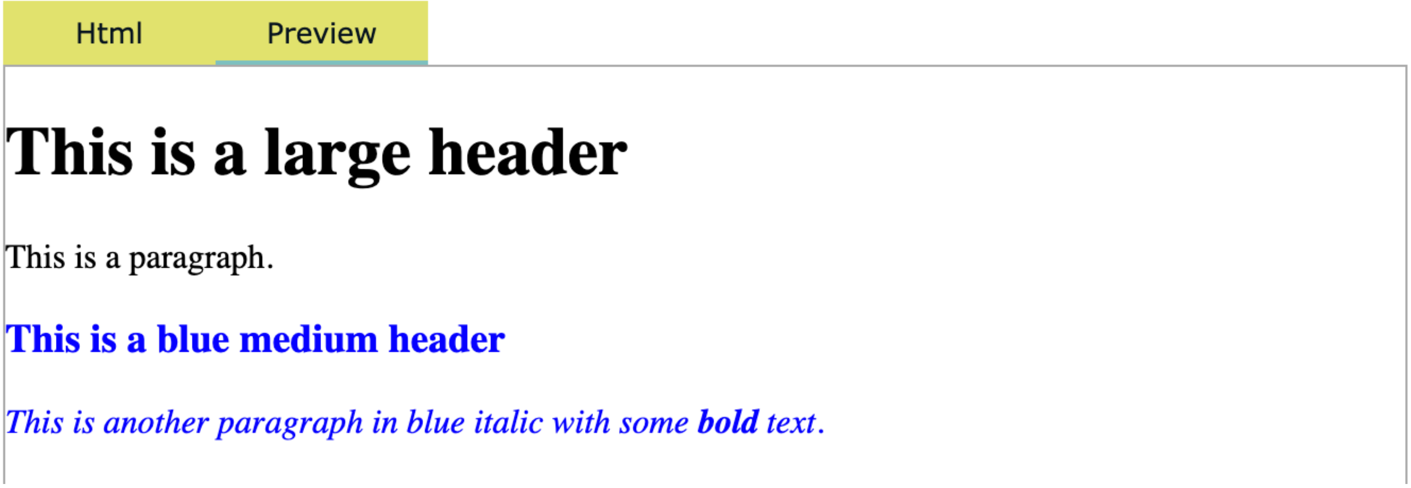
```
<html>
  <body>
    <h1>This is a large header</h1>
    <p>This is a paragraph.</p>
    <div style="color:#0000FF">
      <h3>This is a blue medium header</h3>
      <p><i>This is another paragraph in blue italic with some <b>bold</b> text.</i></p>
    </div>
  </body>
</html>
```

Figure 176:

The same app contains a more complex example showing how you can format HTML for an email. In addition to the HTML example app, there are also two examples in the **Samples** section in the **Hub** (and in the Component Gallery) to display a **Twitter button** and a **Twitter Timeline** that use the HTML component to embed the necessary HTML into a remote form.

The following HTML is used to create the initial Twitter button:

```
<div><a href="https://twitter.com/intent/tweet?screen_name=omnisstudio&text=I love Omnis" class="twitter-menti
```



This is a large header

This is a paragraph.

This is a blue medium header

*This is another paragraph in blue italic with some **bold** text.*

Figure 177:

Figure 178:

Building an HTML block

You can use the *Text:* or *Sta:* command to build up the HTML content. The following method constructs some HTML and assigns it to $html of an HTML control called *HTML*.

```
# lHTML is a local var of Character type
Begin statement
#  ===== Styles =====
Sta: <style>
Sta: p {color: #00F; margin: 0 2em; padding: 1em; background-color: #fff; border: #DDD solid 2px;}
Sta: h1, h2 { color: #666; margin-left: 0.5em;}
Sta: img {margin: 5px 0 5px 50px;}
Sta: </style>
#  ===== Content =====
Sta: <div>
Sta: <h1>HTML Control</h1>
Sta: <p>
Sta: You can display HTML in the HTML control.<br />
Sta: This is the Second line of content.<br /><br />
Sta: <i>And you can do italic etc!</i>
Sta: </p>
Sta: <h2>Pictures</h2>
Sta: <p>You can embed pictures.</p>
Sta: <img src="/images/cat.jpg" width="200" height="176" alt="cat" />
Sta: </div>
End statement
Get statement lHTML
# the html form object is called HTML
Calculate $cinst.$objs.HTML.$html as lHTML
```

You can embed any of the standard HTML tags into the $html property, including links, styles, and tables. The following text could be added to an HTML control which is placed next to a check box control on a remote form to allow end users to popup a window containing competition rules that are stored in a static html page:

```
<p>I have read and agree to the <a href="rules.html" target="_blank">competition rules</a></p>
```

Events

The HTML Control reports the evBefore and evAfter events, as well as evClick and evDoubleClick.

Style and Attribute Placeholders

You can inherit the effects and font attributes that you set for the HTML object in the Property Manager in design mode by inserting various placeholders in the $html property. The placeholders take the format %<letter>, for example, insert %f to inherit the font specified in $font, or %t to inherit the text color set in $textcolor. Most of the placeholders should be inserted into an html element using the style tag, while others are attributes and can be inserted directly into an HTML element.

Placeholder	Type	Description	Property or desc
b	attribute	back color and alpha	$backcolor and $alpha
f	style	font	$font
z	style	font size	$fontsize
s	style	font style	$fontstyle
j	style	font align	$align
t	style	text color	$textcolor
e	attribute	effect	$effect
p	style	position	coordinates of omnisobject
v	style	vert scroll	$vertscroll must be kTrue
h	style	horz scroll	$horzscroll must be kTrue

For example, you can set $html to <div %e></div> and the text in the control will take on the effect from $effect (or $linestyle or $bordercolor, as these are effect properties).

To use the font specified in the HTML object in a paragraph in your HTML insert: <div><p style="%f">SOME TEXT</p></div>. For example, the following HTML produces some text rendered in plain or default HTML paragraph style and second paragraph of text that uses the font setting specified in $font in the HTML object, which in this case is set to "Verdana,Arial,Helvetica,Sans-serif":

```
<div><p>PLAIN TEXT</p><p style="%f">STYLED TEXT</p></div>
```

HTML object in Paged Panes and Subforms

If the HTML object appears in a Paged Pane or Subform (i.e. inside any container field), you will need to add the "white-space:normal" parameter to the <div> tag containing your Html to allow the text to wrap correctly inside the object. For example, the following <div> tag will contain text etc within a 300px width:

```
<div style="width:300px; white-space:normal">
# your HTML
</div>
```

Hyperlink Control

Group	Icon	Name	Description
Navigation	LINK LINK LINK	Hyperlink	List containing hyperlink style opti

The **Hyperlink Control** allows you to present a list of options to the end user, where each option in the list is displayed as a web-style hyperlink; each link option corresponds to a line in the underlying list variable used to populate the list. If the whole list does not fit inside the control, the list will "pop up" when the end user passes the mouse over the control. The $extraspace property specifies extra spacing between links. The Hyperlink control is used in the Studio Browser and is very similar to the JavaScript version of the control.

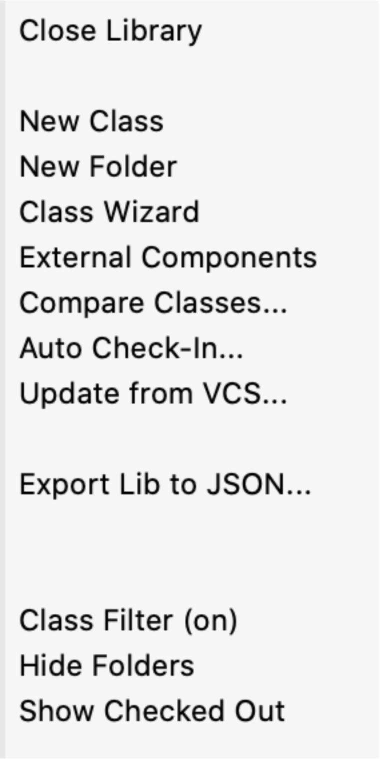


Figure 179:

The options or list contents in the Hyperlink control are based on the contents of a *list variable* specified in $dataname. The first column in the list should be the text displayed for each option listed in the control, unless $listcolumn is specified which is the column of the list variable used to populate the control. Subsequent columns in the list can be used to specify other values.

When the end user clicks on the list an evClick is triggered with the selected line reported in pLineNumber. You can use the value of pLineNumber in your event method to trigger an action. You can create an empty line by putting ”-“ (hyphen) in the first column.

If $isvertical is kTrue the list pops up vertically, and if $shouldunderline is kTrue the links are underlined. The $selectedtextcolor property specifies the color of the text when the mouse is over it.

Example

There is an example library called **JS Hyperlink** in the **Samples** section in the **Hub** in the Studio Browser showing how you can build a dynamic hierarchical list using the Hyperlink control; the same app is available in the JavaScript Component Gallery.

The following method is the $event() method for the Hyperlink control:

```
# iHyperLinks is a List var defined in the construct of the form as
# Do iHyperLinks.$define(iHyperName,iHyperGrp,iHyperCmd)
On evClick
  Do iHyperLinks.$line.$assign(pLineNumber)
  Do iHyperLinks.$loadcols()

  If iHyperGrp=0
    Do method getHyperLinks (iHyperCmd)
  Else
    If iHyperCmd=99
      Do method getHyperLinks (0)
    Else
      Calculate iMsg as con('HyperLink ',iHyperCmd,' of Group ',pick(iHyperGrp,'','A','B','C','D'),' Clicked')
    End If
  End If
```

Label Object

Group	Icon	Name	Description
Labels	T	Label Object	Basic label object

The **Label Object** lets you add standard text labels to your remote form. The text for the label is entered into the $text property in the Property Manager, or you can double-click on the text for the label to edit it. Alternatively, you can assign some text to the $text property in your code.

The font and alignment of the label is setup under the Text tab in the Property Manager. You can use the style() function to style parts of the text assigned to $text, for example:

```
Calculate $cinst.$objs.label1.$text as con('Mellow ',style(kEscColor,kYellow),"Yellow")
```

Styled text is only supported if $textishtml is false. When set to true, the $textishtml property means the text in $text is interpreted as HTML, so in this case you can use any HTML tags and styling to format the text. For example:

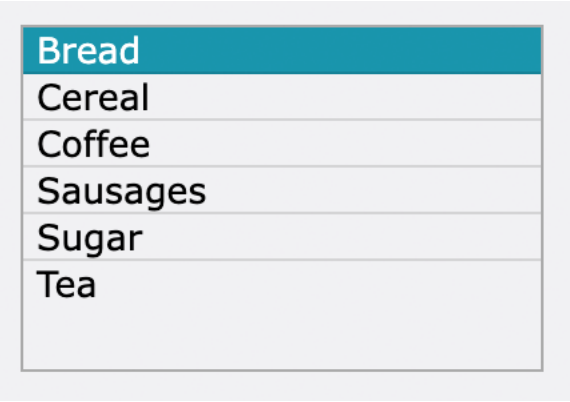
```
Calculate $cinst.$objs.labelhtml1.$text as 'This is <span style="color:#ff0000;">RED</span>'
```

Labels have the $dataname property which means you can assign the text from a variable. In this case, the value in $dataname overrides the value in the $text property, and actually sets $text. You can assign a Date variable to $dataname, in which case the label takes its display format from #FDT, #FD, or #FT depending on whether the variable has a date/time component.

List Control

Group	Icon	Name	Description
Lists	☰	List	Standard list field for displaying list data

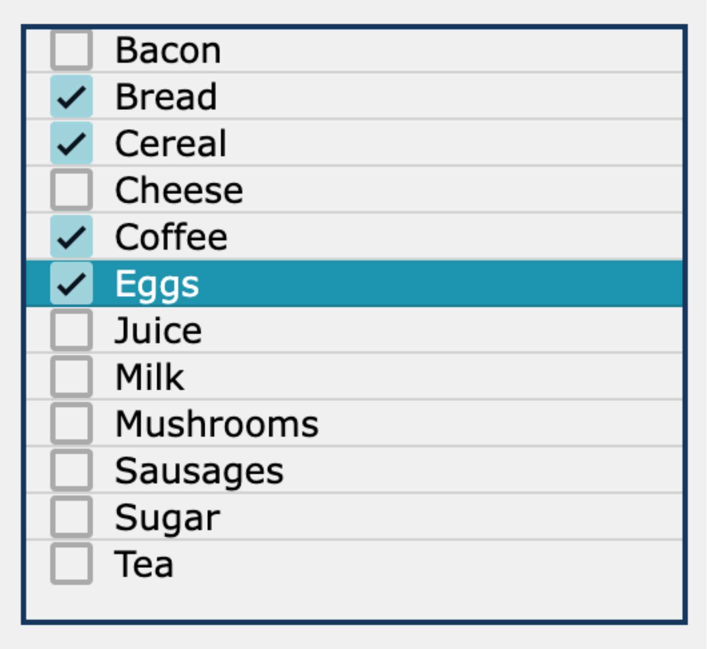
The standard **List Control** allows you to display a single column list on your remote form allowing end users to select a particular line; the following screen shows a simple product list using the 'ice' JS Theme.



Bread
Cereal
Coffee
Sausages
Sugar
Tea

Figure 180:

If the $ischecklist property is kTrue in the standard List control, each list line has a checkbox which allows the end user to select or de-select the line, which is useful for multi-select lists or checklists.



☐ Bacon
☑ Bread
☑ Cereal
☐ Cheese
☑ Coffee
☑ Eggs
☐ Juice
☐ Milk
☐ Mushrooms
☐ Sausages
☐ Sugar
☐ Tea

Figure 181:

There are several example apps showing different types of list control in the **Samples** section in the **Hub** in the Studio Browser (see **JS List, JS Linked List, JS List Pager**), and the same apps are available in the JavaScript Component Gallery.

For lists with many columns, you may prefer to use a Data Grid Control, or for a more compact single-column list you can use a Droplist Control. If you want to include an icon next to the text in each list line you can use a Tree List Control by including your data and icon references at the top-level of the list data without using any child data. For a further alternative to the standard List box, you can use the Native List to display list data but with a native appearance on each OS.

List data variable

The contents of a standard List is taken from the instance variable (of List type) specified in the **$dataname** of the control. In this case, the first column of the list contents is used to populate the list control; you can specify an alternative column for the data in $listcolumn. You can build the list using an instance variable of List type in the $construct method of the remote form so it is available to all instances; the list can be built from your database or from a number of static values.

The **JS List** example app in the Hub uses a simple check list ($ischecklist = kTrue) and a standard List control; the check list is built in the $construct method of the form, as follows:

```
## iCheckList: instance var of List type
Do iCheckList.$define(iCheckListValue)  ## defines the list
Do iCheckList.$add('Milk')              ## add the values
Do iCheckList.$add('Tea')
Do iCheckList.$add('Coffee')
Do iCheckList.$add('Sugar')
Do iCheckList.$sort(iCheckListValue)    ## sorts the list
# in addition you can select a line if required
Do iListvar.$line.$assign(1)            ## selects the first line
```

The example app provides a button to merge the selected lines in the check list into the other list. The code behind the button is:

```
On evClick
  Do iList.$merge(iCheckList,kFalse,kTrue,kTrue)
  # merges the selected lines only
```

See the ‘Programming Lists’ chapter in the *Omnis Programming* manual for more information about using list variables in your code.

Selected Line Color

You can control the color for selected lines by setting $selectedlinecolor; use kColorDefault for the default selected line color for the client OS, or the current JS theme. The $evenrowcolor property sets the background color of even numbered rows displayed in the list; kColorDefault means use the same color as the odd numbered rows ($backcolor).

List Events

When the user selects a line the evClick event is reported with the pLineNumber parameter reporting the selected line number. You can also set the evDoubleClick event for a list control to detect when the end user double clicks on a list line. List events will retuen the pLineNumber parameter containing the number of the line clicked, and you can use this in your event method behind the list to trigger an action.

A double-click event is sent to a List control if the Enter or Space key is pressed while the focus is in the List *and* if evDoubleClick is enabled on the list. Otherwise, if the control has an evClick event enabled, Enter/Space sends an evClick. If the control does not have the evClick or evDoubleClick enabled, then Enter triggers the okkeyobject (if there is one), as long as the state of the list has not changed, i.e. the current line has not changed, and no checkbox has been toggled.

Lists and Client Methods

Searching Lists

You need to prefix the column name with $ref for a search to work in a client method. For example:

```
Do iList.$search($ref.iCol="ABC")
```

Smart Lists and the JavaScript Client

The JavaScript Client does not support smart lists in client executed methods, insofar as if you change the list in some way on the client, it will no longer be a smart list when the updated data is sent from the client back to the server. Smart lists work as expected in server executed methods.

List Scrolling

Lists have the $vscroll and $hscroll properties which allow you to scroll a list vertically or horizontally at runtime in the client browser; note these properties are write-only meaning that you cannot return their values at runtime. The vertical or horizontal scroll value assigned using $vscroll or $hscroll is the row number for a list.

Changing Current Line from the Keyboard

The $keyboardchangesline property determines whether or not the *the current line* in the list changes when the end user navigates the list using the keyboard; the property is available in the standard JS List, Native lists, Tree lists, and Data grids, and only takes effect when $multipleselect is kTrue. This allows multiple rows to be selected while also having the keyboard change the current line. To enable users to select non-adjacent lines with the keyboard, $keyboardchangesline can be set to kFalse.

When the list only allows a single line to be selected ($multipleselect is kFalse), navigating the list with the keyboard always uses a focus line and the user has to manually select a current line with the Space or Enter key, at which point evClick is fired.

Linked Lists

You can link a List control to an Entry field to create a “Linked List”, so that when the end user types into the edit field the contents of the list is updated dynamically. There is an example app called **JS Linked List** in the **Samples** section in the **Hub** in the Studio Browser to show the use of a Linked list; the same app is available in the JavaScript Component Gallery.

The combination of an Entry field with key presses enabled and a List control allow you to create a dynamic list that has the ability to update in response to what the end user types into the edit box. A Linked List is in effect like a Combo box, but with the extra ability to update itself as the user types. To enable this feature, edit controls can detect certain key presses and can be linked to a list control, while for lists themselves there is a property to display selected lines only.

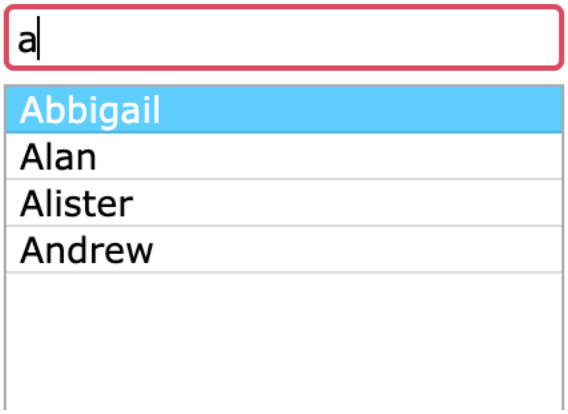


Figure 182:

Creating Linked Lists

The Edit Control has the $linkedobject property which is the name of a list control on the current remote form used to display suggestions to the user if the Entry field enables the evKeyPress event. You need to add the code to populate the list in response to evKeyPress, based on the current value of the instance variable specified for the Entry field.

When the end user types into the Entry field, the linked list will appear automatically. It is not necessary to process any events for the list control, since the dynamic list behavior is determined automatically by being linked in this way. Therefore, you would expect the following to happen when end users are using a dynamic linked list:

- Clicking on a line in the open list will set the edit control to the value of the clicked line.
- Pressing up or down arrow when the focus is on the list control will set the value of the edit control to the new current line of the list.
- Pressing the down or up arrow when the edit control has the focus and when the list is not visible will open the list, set the edit control value to the selected line in the list, and move focus to the list.
- Typing into the edit control when the list is closed will open the list and keep focus on the edit control.
- All the following will close the list: pressing return when the list has focus; pressing escape, or clicking away from the list or the edit control.

Optimizing the Linked List

The list property $selectedlinesonly specifies that only the selected lines in a list will be displayed (this only applies when $ischecklist is false), which in the context of Linked lists allows you to filter the content of a list that is linked to an Entry field. You should optimize the $event method for the edit control in the following way:

- Make the $event method for the Entry field execute on the client.
- On evKeyPress, if the list of suggestions is empty (or no longer suitable for the data), call a server method to build the list (with lines selected based on the value of the edit control).
- In subsequent evKeyPress events that can use the pre-built list, perform a list $search to change the selected lines to the ones which are suitable for the new value of the edit control.

In this way, the list of suggestions is cached on the client, and updates simply by changing the selected lines with a search in your code.

When $selectedlinesonly is true, the processing involving the usual click events and so on all use the line number of the *data* in the list, not the lines displayed in the list.

Note that if you use the $evenrowcolor property when $selectedlinesonly is true, the even row color applies to the numbers obtained by counting the displayed lines, rather than using the line numbers of the data in the list.

Detecting Key Presses

Entry fields have an event called evKeyPress; the Key press detection was added to support Linked Lists, but it could be used by itself for other purposes. The evKeyPress event has a single event parameter, pKeyList, which is a *three column list* containing the keys entered since the last evKeyPress event (or since typing started) with a row for each key in the order the keys were pressed.

Each key in the list is either a data character or one of a limited set of system keys (note that not all keyboard keys are supported, such as function keys are not supported). Transitions in shift state, ctrl state, and so on, do not result in a key in the list, so if the user types Shift+a, the character in the list will be A. The supported system keys are:

- Backspace
- Tab
- Escape
- Insert and Delete
- Up, Down, Left, and Right arrow key
- Page Up and Page Down
- Home and End
- Return

The columns in the list are as follows:

- **Colum 1:** If the key is a system key, column 1 is the keyboard constant value representing the key, such as kEscape. Otherwise column 1 is #NULL.
- **Colum 2:** If the key is a data key, column 2 is the character data for the key. Otherwise column 1 is #NULL.
- **Colum 3:** Is the sum of the following keyboard modifier constants, representing the state of these modifiers at the point the key was added to the list: kJSModShift, kJSModCtrl, kJSModAltOrOption, kJSModCmd.

In addition, there are some properties which control when evKeyPress events are generated. The entry field property $keyeventdelay is the minimum number of milliseconds (0-2000) between evKeyPress events. The first evKeyPress will also be delayed for this duration. This allows you to throttle keyboard events in the case where they will be executed on the server, and therefore reduce the load on the server. If true, the $systemkeys property specifies that evKeyPress events include system keys which do not change the value of the data. If false, only system keys such as backspace are included as they potentially change the data.

While the user is typing, the edit control in the user interface remains enabled (unlike normal event processing where the entire user interface is usually disabled). The value of the instance variable associated with the edit control reflects the current content of the entry field when evKeyPress is generated.

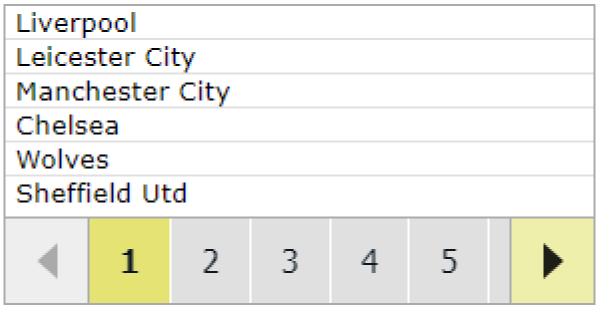
The following method is the $event() method for the Edit field linked to a list of names (from the example app in the Hub):

```
On evKeyPress
  Calculate iKeyList as pKeyList
  Do iLinkedList.$search(mid(low(iLinkedListCol),1,len(iEditVar))=low(iEditVar))
```

List Pager

The **$pagesize** property allows you to display the lines in a list or data grid as separate pages, to improve the user experience when navigating lists or grids with a large number of lines; the default value is zero which means no list pager is displayed. The $pagesize property is available for the JS **List, Data Grid, Complex Grid,** and **Native List** (when not using grouped lists). There is an example app showing the List pager for all these list types in the **Samples** section in the **Hub** in the Studio Browser.

When assigned an integer value, the $pagesize property forces the contents of the list or grid to be sub-divided into a number of scrollable pages, and a set of page number buttons (as well as forward and back buttons) are displayed under the list or grid box, which allows the end user to "page through" each group of lines in the list or grid.



Liverpool
Leicester City
Manchester City
Chelsea
Wolves
Sheffield Utd

Figure 183:

The value assigned to $pagesize specifies the number of list lines displayed in each page, and therefore the total number of lines in the list, divided by the value of $pagesize determines the number of pages in the list or grid field, as well as the number of buttons displayed in the pager panel. The $pagerpage property allows you to set the page to be displayed, with the first page as the default.

Note that setting $pagesize does not reduce the amount of data sent to or fetched from the server – the full list data is sent to the client, and the setting of $pagesize is only used for displaying the list or grid with the pager element.

End users can *long press* on the Previous or Next arrow buttons on a List pager to jump to the *Start* or *End* of the pages displayed in the list control. The timer for the long press is hard coded to 700ms.

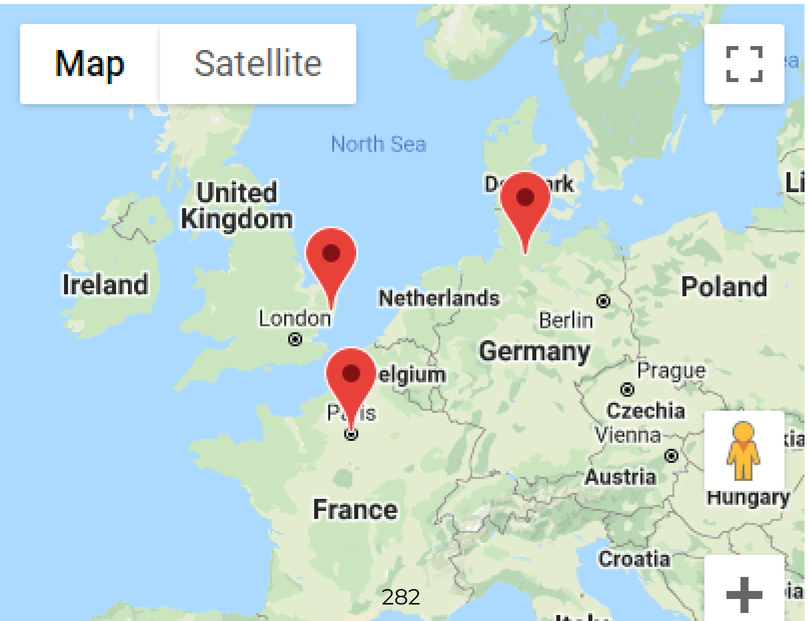
Changing the Pager’s Appearance

The appearance of the pager, such as the color of the buttons, numbers, and arrows, cannot be controlled using standard component properties. However, if you wish to customize the appearance of the pager, you can do so by overriding the associated CSS classes. These classes are named ‘omnis-pager<-xxx>’ and are listed in the core.css file. Note that this file is minified, so you may want to use a prettifier tool to make it more human-legible. Do not edit the classes in core.css, rather you should override the classes by adding your own version in the user.css file found in the html\css folder in your Omnis development tree.

Map Control

Group	Icon	Name	Description
Other	📍	Map	Displays a Google map for specifie location(s)

The **Map Control** allows you to place a Google Map® on your remote form; this provides many of the functions available in a Google map as provided by the Google Maps API, such as placing your own markers or customizing the markers.



There is an example app called **JS Map** in the **Samples** section in the **Hub** in the Studio Browser showing the full capability of the Map control; the same app is available in the JavaScript Component Gallery which you can view online, to see many of its capabilities, but you will need a Google API key to run the example app in development mode.

The JS Map sample app prompts you to enter a Google Map API key when it is opened. The Map API key you enter is stored in the "mapsApiKey" entry in "jsclient" section of config.json and is loaded each time the sample app is opened. When you deploy your app you can add your Map API key to config.json on the Omnis Server.

The Map control has the standard properties to control its appearance including $edgefloat, for example, so you can fit the map to the current client area or container, as well as $borderradius allowing you to set a border radius.

Activation and API Key

In order to use the Map control, you need to register with Google Cloud Platform to obtain a Map API key, which can be entered in the $apikey property of the Omnis Map control; the Map control will not work without the API key. Specifically, you need to:

- Create and activate a Google Cloud Platform account (a free trial is available with credits)
- In the console under Credentials, create a new API key; copy this API key and add it to the $apikey property of the Omnis Map control in your remote form
- Under APIs, navigate to Map APIs and enable the "Maps JavaScript API". In addition, if you want to use the Geocode function, you need to enable "Geocoding API" (there are other APIs which you may be able to use).

Important Note: By signing up to Google Maps API via the Google Cloud Platform and using Google Maps in your deployed Omnis application, you and your end users must agree to the general "Terms of Use" for Google Maps: there is a link to these Terms of use on the map displayed in the Map control. *Omnis Software cannot be held responsible for any third-party products or services and will not be liable for any damage or loss resulting from your use of the Google Maps content or the products.*

Apart from the $apikey property, there is no further configuration needed to display a basic Google map in your remote form, however, to make the map useful you will probably want to display specific locations, add markers on the map, or allow end users to interact with the map: all of these are possible with the JS Map control and are described below.

Map Location and Zoom Level

The $latlong property allows you to specify the center location of the map as "latitude:longitude", for example, a value of 40.749305:-73.985775 will center the map on New York. If you do not set $latlong initially (by setting the property in the Property Manager in design mode or in the $construct method of the form), the map will open at some unspecified location, usually the default location setup in Google on the client device.

The $::zoom property lets you set the zoom level of the map: the range is 0 to 21, with zero showing the largest area (the whole world) and 21 the smallest possible area. The default zoom level is 8 which shows a reasonable amount of detail for the current center location of the map.

Finding the Latitude:Longitude

The Map component supports Google Geocoding which allows you to convert a street address into a geographic coordinate like latitude and longitude, which you can use to place a marker on a map, or center the map.

To use the Geocoding function, you need to access the Geocoding API which itself requires a separate API Key, in addition to the Maps API key, which you can obtain from Google.

A Search button has been added to the JS Map example available in the Hub to show how you can convert a street address to a latitude:longitude coordinate which can be applied to the $latlong map property. Note the example app places the Geocoding API key in the $userinfo property which is then sent to Google.

Alternatively, you can find the latitude:longitude coordinate manually. To do this, Right-click somewhere on a standard browser-based Google map (not the Omnis map control), select the 'What's here' option and the latitude:longitude value of that position is shown on the popup. You need to replace the comma with a colon to be used as a parameter in Omnis, e.g. 52.223460:1.492379.

Map Type and Controls

The $maptype property lets you set the type of map – this can be set to a constant: kJSMapTypeRoad (default), kJSMapTypeSatellite, kJSMapTypeHybrid, and kJSMapTypeTerrain. The end user can change the map type using the map type control shown on the map, assuming the $maptypecontrol property is enabled.

The other standard map controls, including the pan control, scale control, and the street view control are enabled using the properties $pancontrol, $scalecontrol, and $streetviewcontrol, respectively (these are all enabled by default): note you can only change these properties at runtime if the map control itself is enabled ($enable = kTrue).

The zoom control can be further controlled by setting the $zoomcontrol property: it can be set to kJSMapZoomOff, kJSMapZoomDefault (the default), kJSMapZoomSmall, and kJSMapZoomLarge. The latter two settings correspond to a simple Plus|Minus button (Small) or the same button with a vertical slider control (Large) for finer adjustment of the map zoom level.

Map Markers

You can place a marker or set of markers on the map by assigning a row or list containing marker information to the $mapmarkers property. For each marker you must define the latitude and longitude of the marker location in the first column of the setup list in the format latitude:longitude (e.g. “40.749305:-73.985775”), and in subsequent columns you can specify the marker title (shown when you hover over the marker), the tag or title for the popup (shown when you click on the marker), and html content for the popup. An optional fifth parameter can specify an icon for the marker to replace the default map marker (see below). If the third and fourth columns are empty for any marker defined in the list, the marker will not popup when clicked. (See the Events section about how to add markers via user clicks.)

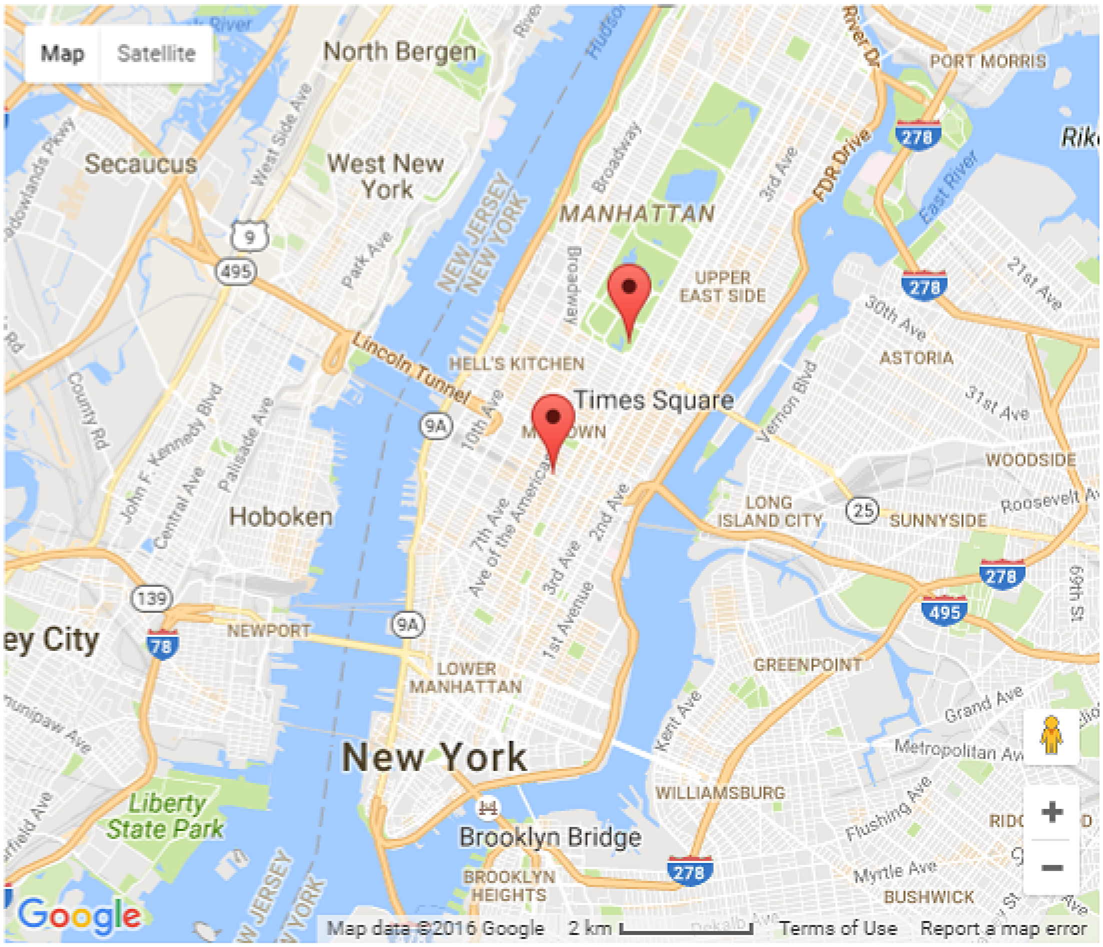


Figure 185:

The following code adds markers to a map indicating the Empire State Building and Central Park in New York, it then centers the map on the Empire State Building, and sets a zoom level of 12 which shows a reasonable level of detail in this case:

```
# create vars map_markers (List), marker_latlong, marker_title,
# marker_tag, marker_html (Chars)
Do map_markers.$define(marker_latlong,marker_title,marker_tag,marker_html)
Do map_markers.$add("40.749305:-73.985775","Empire State Building","Empire State Building",>"<div id='content'
Do map_markers.$add("40.766225:-73.972514","Central Park","NY Central Park","<div id='content'><h3 id='firstHe
Do $cinst.$objs.map.$mapmarkers.$assign("map_markers")
Do $cinst.$objs.map.$latlong.$assign("40.749305:-73.985775")
Do $cinst.$objs.map.$::zoom.$assign(12)
```

Map Marker Icons

The marker list assigned to $mapmarkers can have an optional fifth column which you can use to specify the icon URL for an image for the map marker. An empty string in this column (or a missing column altogether) means that the marker will use the default marker icon. You can use an SVG icon for a map marker in which case it will scale as the map is scaled.

The value for the marker icon is an icon URL which you set using the iconurl() function. Since the marker image has to be set for each row in your list you can specify a different image for each marker in the marker list, but if you want the same image for each map marker you have to set the marker image for every row in your marker list.

When an SVG icon is used, you can add an additional column to the list to specify the color to apply to that SVG icon, which must be themed using the JS Themer tool. The color should either be a JS Theme constant, such as kJSThemeColorPrimary, or an RGB integer.

Custom Markers

You can assign an alternative marker icon or symbol, including map markers from the Google maps API, by adding a sixth column to the marker list: in this case the fifth column should be omitted.

The definition for the markers list in the JavaScript Map control can be:

```
Do iMapMarkers.$define(iMarkerLatLong, iMarkerTitle, iMarkerTag, iMarkerHtml, iMarkerIcon, iMarkerCustom)
```

where *iMarkerCustom* is a new string column (column 6) specifying a custom marker. When a marker is defined in the marker list, and the iMarkerIcon (column 5) is empty, iMarkerCustom can be included with the following attributes, separated with a '|' character (you only need to specify the attributes required). An example custom string would be:

```
"path:google.maps.SymbolPath.BACKWARD_CLOSED_ARROW | fillColor: red | fillOpacity:0.8 | scale: 4 | strokeColor
```

Or to draw a five-pointed star marker:

```
"path:M 125,5 155,90 245,90 175,145 200,230 125,180 50,230 75,145 5,90 95,90 z | fillColor: red |
fillOpacity:0.8 |  scale: 0.1|strokeColor:black | strokeWeight: 1 | anchor:122,115"
```

Or to draw a circle marker:

```
"path:google.maps.SymbolPath.CIRCLE | fillColor: red | fillOpacity:0.8 | scale: 4"
```

Where the custom marker parameters are defined as:

- **path** can either be a map symbol, or an SVG notation path, as defined below
- **fillColor** the color used to fill the marker object, an html css color name or value e.g. #FF0000
- **fillOpacity** the opacity of the fill color, a value from 0 to 1, e.g. 0.5 is 50% transparent fill
- **scale** a scaling factor for the object
- **strokeColor** the color used to outline the object, an html css color name or value e.g. #FF0000
- **strokeWeight** the thickness of the stroke line
- **anchor** allows you to set the anchor position or offset the shape. By default, shapes are aligned to the top left of the marker relative to its lat:long

Marker Symbol	Name – prefixed google.maps.SymbolPath.	Descriptior
○	CIRCLE	A circle
∀	BACKWARD_CLOSED_ARROW	A backwar all sides
A	FORWARD_CLOSED_ARROW	A forward-p sides
∨	BACKWARD_OPEN_ARROW	A backwarc one side
∧	FORWARD_OPEN_ARROW	A forward-p side

For example:

```
Do iMapMarkers.$define(iMarkerLatLong,iMarkerTitle,iMarkerTag,iMarkerHtml, ,iMarkerCustom)
Do iMapMarkers.$add("52.223460:1.492379","Omnis UK","Omnis UK","","","path:google.maps.SymbolPath.BACKWARD_CLO
# the JS Map app uses similar code to show the Omnis offices
```

The **Map** example app includes the use of markers and polygons, and is available in the JavaScript Component Gallery. The following image shows the location of the European Omnis offices using the “Backward-pointing Closed Arrow”.

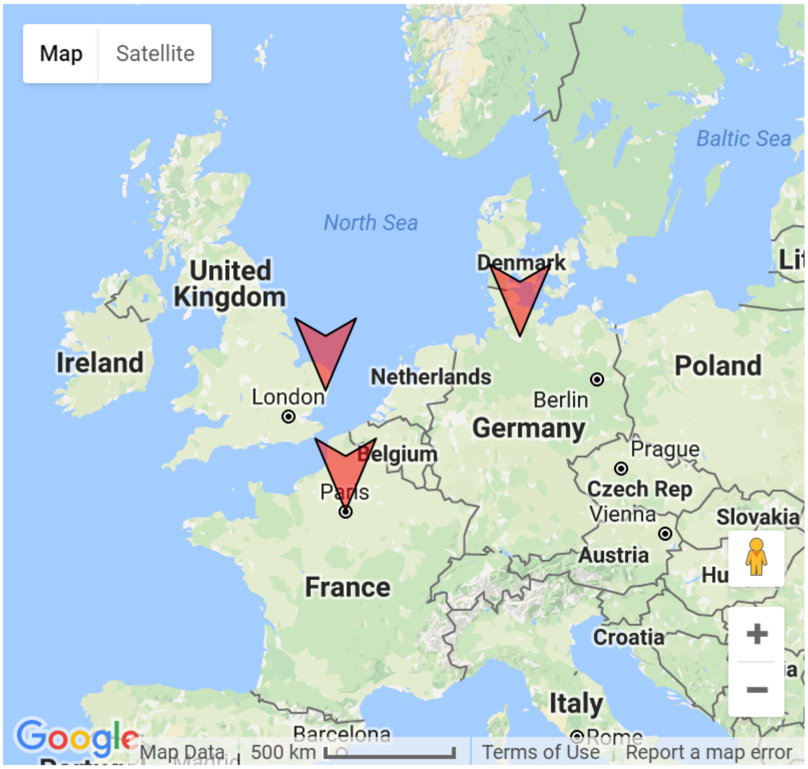


Figure 186:

The map control has a property $fitmaptomarkers that can be assigned value 1 at runtime to force the map to zoom in or out to allow all the map markers to be shown.

Polygon Objects

In addition to icons and standard map markers, you can add polygon objects or irregular shapes to maps in the JavaScript Map control. The $mappolys property specifies the data name of a list which contains the definition of each polygon or shape as follows:

```
Do iPolyMarkers.$define(iPolyLatLong,iPolyStroke,iPolyOpacity,iPolyWeight,iPolyFill,iPolyFillOpacity,iPolyTag)
```

- **iPolyLatLong** the latitude:longitude values for each of the the points of the polygon, so a triangle would have 3 points: the lat:long settings are separated with the '|' character, e.g. 25.774,-80.190|18.466,-66.118|32.321,-64.757|25.774,-80.190
- **iPolyStroke** the color used to outline the polygon, which is an html css color name or value e.g. #FF0000
- **iPolyOpacity** the opacity of the stroke color, a value from 0 to 1, e.g. 0.5 is 50% transparent
- **iPolyWeight** the thickness of the stroke line
- **iPolyFill** the fill color of the polygon object, an html css color name or value e.g. #FF0000
- **iPolyFillOpacity** the opacity of the fill color, a value from 0 to 1, e.g. 0.5 is 50% transparent
- **iPolyTag** the tag name or label for the polygon, which is sent to the evPolygonClicked event method in pPoly

For example, the following code draws the Bermuda Triangle on the map (see the JS Map example app):

```
Do iPolyMarkers.$add("25.774,-80.190|18.466,-66.118|32.321,-64.757|25.774,-80.190","#FF0000","0.8","3","#FF000
```

The evPolygonClicked event with the parameter pPoly is called when a polygon on the map is clicked, and pPoly will be set to the polygon tag as defined in the list.

Events

There are various events available in the map control to allow you to detect when and where the map was clicked (evMapClicked, reports latitude and longitude of the click), when the map is dragged by the end user (evMapMoved, reports the latitude and longitude of the new center location), when the map is zoomed using the zoom control (evMapZoomed), or when a map marker is clicked (evMarkerClicked). None of these events are enabled by default, so you have to enable them in the $events property for the control using the Property Manager.

The following $event method could be placed behind a map control to detect when the map is clicked, or when a marker is clicked (assuming a marker has been added), or if the map is moved or zoomed. The add_markers Boolean variable is linked to a checkbox on the window to allow the end user to enable or disable the ability to add markers. When the map is clicked, the evMapClicked event reports the position in pLatlong, and a marker definition is added to the map_markers list. The message variable is assigned to a field on the form to show either the new center of the map, the new zoom level, or which marker has been clicked.

```
# $event method for map control
# define vars map_markers (List), marker_latlong, marker_title,
# marker_tag, marker_html, message (Chars), add_markers (Bool),
# marker_no (Long int = 0)
On evMapClicked
  If add_markers  ## linked to checkbox on window
    Calculate marker_no as marker_no+1
    Do map_markers.$define(marker_latlong,marker_title,marker_tag,marker_html)
    Do map_markers.$add(pLatlong,con("Marker ",marker_no),con("Marker ",marker_no),con("This is marker ",marke
    Do $cinst.$objs.map.$mapmarkers.$assign("map_markers")
  End If
  Calculate message as con(pick(add_markers,"Map clicked here: ","Marker added here: "),pLatlong)
On evMapMoved
  Calculate message as con("Center is now: ",pNewCenter)
On evMapZoomed
  Calculate message as con("New zoom level: ",pNewZoom)
On evMarkerClicked
  Calculate message as con("Marker clicked: ",pMarker)
```

Native List

Group	Icon	Name	Description
Native	☰	Native List	List control with platform depende appearance

About the Native Controls

The **Native** group in the Component Store contains components that have a more familiar or “native” appearance when they are displayed on different mobile platforms, that is, iOS and Android – their appearance is rendered in the JavaScript Client using CSS customized for each platform. The different appearance for each platform is handled by Omnis automatically, therefore you only need to setup the component once in design mode. See also: Native Slider and Native Switch.

When running on a supported device, these controls will render and work in a manner close to a device’s native versions. For example, a Native Switch control will look like an iOS switch on an iOS device, an Android switch on an Android device, and so on, while using a single control and set of methods in design mode in your remote form.

JavaScript Remote forms have an Appearance property called $defaultappearance which determines both how a native control is displayed in the design window, and how it would render on non-supported clients (e.g. Desktop browsers). The property is either kAppearanceiOS or kAppearanceAndroid (kAppearanceBlackberry is no longer supported).

The $defaultappearance property can also be switched using the ‘Native Components Display As’ context menu option of a JavaScript Remote Form (right-click on the form to open the context menu). You can also cycle through the values using the keyboard shortcut Ctrl-Shift-N on Windows or Cmd-Shift-N on macOS when the remote form is the top window.

It is recommended that you set $disablesystemfocus property to kTrue for any native controls you have used, to prevent the focus ring being drawn around a controls when it is selected – otherwise the focus ring may interfere with the native appearance.

Creating a Native List

The **Native List** control provides a list control that has a native appearance on different platforms, that is, it has a different appearance on iOS and Android. You can customize the accessories in the list by adding your own HTML content and CSS styling, and therefore provide a very rich UI in a single list format.

In keeping with this philosophy, the JS Native List exposes many appearance properties to allow you to customize the list how you wish. If you leave any of the values as kColorDefault, they will revert to the platform-specific default.

The native list has been designed along the same lines as the iTableView component of iOS forms. The list can either be a standard flat list or a grouped list, using nested lists.

To increase efficiency, the list only draws the displayed rows, along with several more in a buffer zone around them, at any one time. This provides smooth scrolling, and means that the size of the list has very little impact on performance. As a side-effect of this, when scrolling quickly, you will see that the rows may not be rendered immediately.

There is an example app called **JS Native List** in the **Samples** section in the **Hub** in the Studio Browser showing how you can customize the accessories in a native list; the same app is available in the JavaScript Component Gallery.

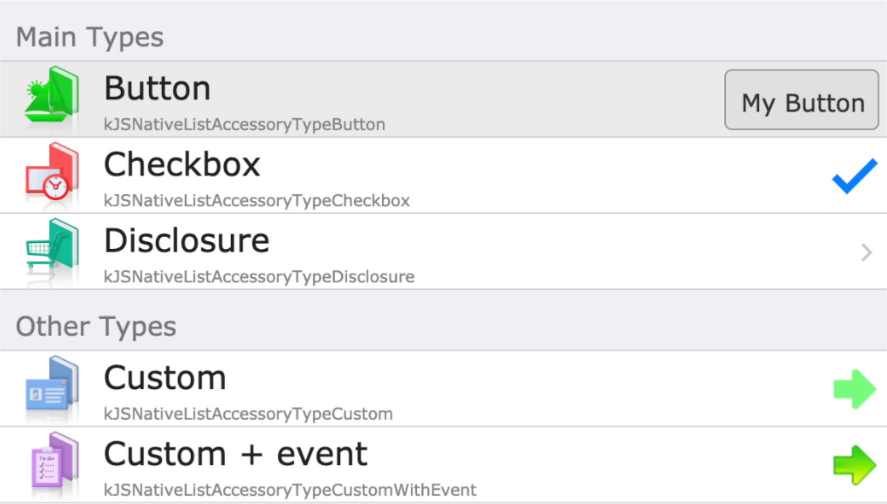
Defining the $dataname list

The structure of the List instance variable used for the $dataname of the native List control differs based on how you wish the list to display.

The **Data** tab in the Property Manager allows you to assign column numbers for each row part, i.e. $text1col allows you to specify which column in your list contains the the data to display as the main text of the row. If you do not wish to make use of a particular row part, leave its column set to 0.

You need to define and populate your list in accordance with the column numbers you have set in the **Data** tab of the Property Manager. The content of these columns should be as follows:

- **text1col:** This should be Character data, to display as the main text for the row.



Main Types

Button
kJSNativeListAccessoryTypeButton
My Button

Checkbox
kJSNativeListAccessoryTypeCheckbox

Disclosure
kJSNativeListAccessoryTypeDisclosure

Other Types

Custom
kJSNativeListAccessoryTypeCustom

Custom + event
kJSNativeListAccessoryTypeCustomWithEvent

Figure 187:

- **text2col:** This should be Character data, to display as the secondary text for the row.
- **imagecol:** This should be Character data - a URL to an image to display.
  The image will be scaled to fit the size of the row's image (customized using $imageheight & $imagewidth).
  The URL can make use of Omnis' support for pixel-density-aware image selection by passing the URL in the format: "<URL to 1x image>;<Name of 1.5x image>;<Name of 2x image>" (where all 3 images are in the same location). This means that on a Retina device it will use the 2x image, but on a standard display device it will use the 1x image. As the image is scaled anyway, you could just always use the 2x image, but this method reduces unnecessary bandwidth usage and processing of larger images
- **accessorytypecol:** This should be a kJSNativeListAccessoryType… integer value. It determines the type of accessory to display on the right edge of the row.
  Use *kJSNativeListAccessoryTypeNone* for no accessory
- **accessoryvaluecol:** Contains the current value of the row's accessory.
  This is currently only used by the Checklist accessory, to represent the checkbox's state
- **accessorycontentcol:** This should be Character data, and should describe the content for some accesssory types. The Accessory types which make use of this are:
  Button: For rows with a button accessory, the content should be the text for the button.
  Custom: For rows with a custom accessory, this should be HTML to describe the contents of your custom accessory.
  CustomWithEvent: see below.

Custom Accessories

You can add your own custom row accessories by setting the **accessorycontentcol** list column to: **kJSNativeListAccessoryTypeCustom, kJSNativeListAccessoryTypeCustomWithEvent** or **kJSNativeListAccessoryTypeMenu** (see below).

kJSNativeListAccessoryTypeCustomWithEvent works in the same way as **kJSNativeListAccessoryTypeCustom** but will trigger a different click event when you click on the accessory (evClick will be called with pWhat equal to **kJSNativeListPartAccessory** instead of **kJSNativeListPartRow**).

When **accessorycontentcol** is set to a custom accessory, its value should be the HTML content of your custom accessory, encapsulated within a single parent element.

You can set your accessory to a particular size (rather than fill the available space in the row) by providing **width** & **height** values in the **style** tag of your top-level HTML element.

If a size is defined in this way, the native list control will attempt to center the accessory. If this does not work, you may also need to set "**position: absolute;**" in your style tag. For example:

```
'<div style="position: absolute; width: 50px; height:25px; background-color: #FF0000;" />'
```

Menu Accessory

The **kJSNativeListAccessoryTypeMenu** accessory adds a menu button to a native list row. The menu can be defined either with the $menulistname property, or a method of the native list called $populatemenu.

The **$menulistname** property can be assigned a list to define the rows in the menu, which will be used for the menu in *all rows* in the native list, unless it is overridden by $populatemenu, in which case, menus can be assigned on a per row basis. The **$populatemenu** method is called when the user selects a menu button which should return a list. The first parameter is the group ID, the second is the row ID. This method can be client or server executed.

Menu lists should have the following columns:

- **Text** (Character): The menu line text
- **Enabled** (Boolean) [optional]: Whether the line is enabled or disabled
- **CommandID** (Integer) [optional]: The command ID
- **BackColor** (Integer) [optional]: The line's background color. Zero means use the default color
- **TextColor** (Integer) [optional]: The line's text color. Zero means use the default color. If lBackColor is a theme constant and lTextColor is null, the text will use the corresponding theme text color

When the user selects a menu line, **evClick** is sent with pWhat=kJSNativeListPartMenuLine. The parameters pMenuLineNumber and pMenuCommandID can be used to identify the line that was clicked.

Menu Icons

You can add icons to menu rows by adding the following columns to the list defining the menu (specified in $menulistname):

- **IconURL** (Character): the icon URL for the line
- **IconColor** (Integer): the icon color for themed SVGs. kColorDefault means the icon will use the menu text color

Creating a Grouped list

If you wish your Native List control to display its data as a Grouped list, you need to change the structure of the list assigned to *$dataname*. The main list should comprise two columns:

- **Column 1:** Should be defined as being of type "List", and each row should contain a list structured as defined above (adhering to the columns specified in the *Data*tab of the property inspector). All of the rows defined in this sub-list become part of a single group.
- **Column 2:** A Character column, defining the name of the group.

Vertical Scrolling

When you set **$vscroll** (to an integer), a Native List will scroll to the specified row number. For grouped lists, group headers are counted as rows, so to scroll to the 5th row of the 2nd group, you would set $vscroll to 1 + [no. rows in group 1] + 1 + 5.

Row Display Style

The **$rowdisplaystyle** property determines how the row is displayed. Its value is a *kJSNativeListDisplayXXX* constant, allowing you to change between displaying the two text fields in a vertical or horizontal orientation.

Date and Number Formatting

The **$dateformat, $dateformatcustom** and **$numberformat** properties allow you to set the date and number format for a Native list.

Reordering Rows

The **$reordermode** property can be set to a kJSReorderMode... constant to specify whether rows in a Native list can be reordered. When enabled, a drag icon is added to each row on the left or the right side of the list to allow you to drag individual rows. The constant values are:

Constant	Description
kJSReorderModeNone	Reordering is disabled (draggable regions are hidden)
kJSReorderModeLeft	Reordering is enabled with draggable regions on the left side of the list
kJSReorderModeRight	Reordering is enabled with draggable regions on the right side of the list

When **$reorderbetweengroups** is set to kTrue (the default), end users are able to drag a row into a different group, otherwise if kFalse, rows can only be dragged within their own group.

The **evReorder** event reports the old and the new position (and group, if applicable) of the row that has been reordered:

- **evReorder**
  Sent when the list is reordered, with the parameters:
  pFromGroup: The old group of the moved row
  pFromRow: The old position of the moved row
  pToGroup: The new group of the moved row
  pToRow: The new position of the moved row

Advanced Customization

If you wish to change the default appearance of the Native list for a particular platform, or wish to change something which is not exposed as a property, it is recommended that you extend any relevant CSS styles in your **user.css** file.

When extending the CSS used by any controls in Omnis, there is the possibility that you may change how a control appears or behaves (especially if you alter sizes), so you do so at your own risk.

Native Slider

Group	Icon	Name	Description
Native		Native Slider	Slider control with platform depen appearance

The **Native Slider** control works, for the most part, in the same way as the standard Slider component but has a more familiar appearance when running on different platforms, that is, it has a different appearance on iOS and Android. There is an example app called **JS Native Slider** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery. See also Native List for general information about using the Native controls.

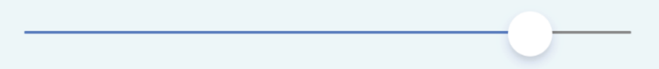


Figure 188:

The current value of the slider is reported in the property $val according to where the slider is positioned. You can specify the range for the slider in the $::min and $::max properties. The $usessteps property is a Boolean determining whether or not the slider should snap to discrete step values specified in $step.

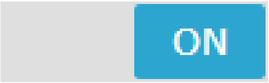
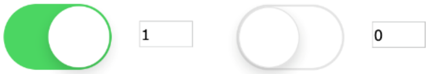
The Slider reports three events: evStartSlider (when the control is starting to track), evEndSlider (when the control has finished tracking), and evNewValue (when the value has changed). You can detect these events in the $event method for the component. These events all pass the current value of the Slider in the pSliderValue parameter. As the user drags the Slider thumb the evNewValue event is triggered and pSliderValue is sent to the $event method for the Slider.

If you do use evNewValue, you should mark your $event method as client-executed and consider enabling the $usessteps property and setting $step to limit the number of events being triggered as the user moves the slider. Alternatively, you could use the evEndSlider event to report the final value since for most purposes this will be the value selected by the user.

Native Switch

Group	Icon	Name	Description
Native		Native Switch	Switch control with platform depe appearance

The **Native Switch** control works, for the most part, in the same way as the standard Switch component but has a more familiar appearance when running on different platforms, that is, it has a different appearance on iOS and Android. There is an example app called **JS Native Switch** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery. See also Native List for general information about using the Native controls.

Native Switch on Android	Native Switch on iOS
	

The Switch has a **$dataname** property, to which you can assign a **Boolean** instance variable. This will be kept up to date with the state of the switch, as the end user clicks or taps on the control.

The Switch has **$justifyhoriz** and **$justifyvert** properties. For some platforms (e.g. iOS) the switch maintains a specific aspect ratio. These properties determine how the switch is positioned inside the control in these circumstances.

The text displayed on the Native Switch is controlled by two members in the built-in strings object jOmnisStrings in the JavaScript Client. You can use the members “switch_on” and “switch_off” to replace the default text with your own text, for example, if you wish to provide different language equivalents to the default text.

Information about how to change or localise these strings can be found in this manual, under “Localizing Built-in Strings”.

If you wish to override the base text, you could use the following code in a separate JavaScript file loaded in your form’s html file after omjsclnt.js:

```
  jOmnisStrings.base.switch_on = "I";
  jOmnisStrings.base.switch_off = "O";
```

Navigation Bar Control

Group	Icon	Name	Description
Navigation		Navigation Bar	Navigation bar with page selectio

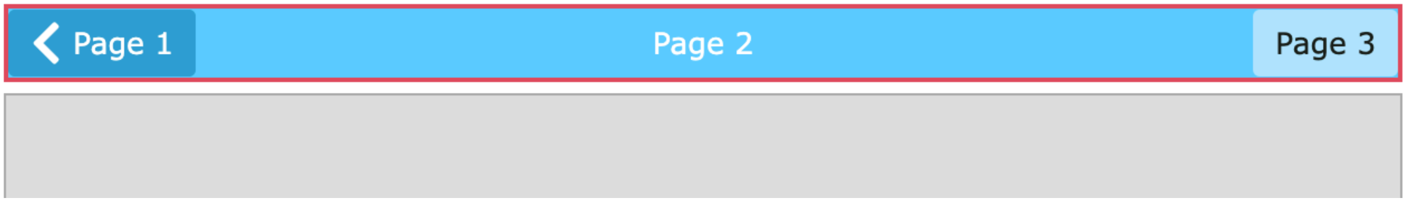


Figure 189:

The **Navigation Bar** control (or Nav Bar) provides a standard navigation bar which end users can use to navigate to different parts of your application. There is an example app called **JS Nav Bar** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery. ; the following screen shows the example Nav Bar using the 'sky' theme.

The Navigation Bar has a main title in the middle of the control, and it can have a left and/or right button which respond(s) to user clicks; see above, Page 2 is displayed, with 'back' and 'forward' buttons displayed on the left and right, respectively.

The Nav bar can be linked to a **Paged Pane** via the **$linkedobject** property to allow you to display different panes in your form in response to clicks in the Nav bar. Actions for the Nav bar can be stacked up using the $push property: see the example below.

The Nav Bar has a number of properties for setting the color, fonts and style of the bar and buttons ($button.. properties), together with the following properties:

Property	Description
$disableanimation	Disables the animation when moving between pages. This property can only be set in design mode (not at runtime using the notation)
$initiallefticonid	If this is not zero, and $initiallefttext is empty, the first navigation bar item has a button on the left hand side, displaying this icon
$initiallefttext	If this is not empty, the first navigation bar item has a button on the left hand side, displaying this text
$initialrighticonid	The icon for the initial navigation bar button on the right
$initialrighttext	The text for the initial navigation bar button on the right
$initialtitle	The initial title displayed on the navigation bar
$lefthidden	If true, the left hand (back) button is hidden for the current navigation bar stack item
$linkedobject	The name a paged pane on the current remote form, used in conjunction with the $push property

Property	Description
$push	At runtime, allows you to assign a 2-6 column row to the paged pane referenced in $linkedobject: col1 is the page number of the paged pane col2 is the title for pushed item col3 is the text for right button (pass empty for no right button) col4 is the or icon id for the image for right button col5 can be non-zero to hide the left button col6 is the text for left or 'Back' button (pass empty to display the title of the previous pane by default). The path to the icon referenced in col4 must be obtained in a server method using the iconurl(icon-id) function since icon ids cannot be resolved on the client
$pop	Removes a specified number of items off the navigation stack: see below
$righticonid	If this is not zero, and $righttext is empty, the current navigation bar item has a button on the right hand side, displaying this icon
$righttext	If this is not empty, the current navigation bar item has a button on the right hand side, displaying this text
$::title	The title for the current navigation bar stack item

Setting $disableanimation to kTrue disables the animation when moving between pages: this property was added to fix a problem when using the built-in VoiceOver screen reader on an iPad in conjunction with a Nav Bar linked to Page Pane: the problem is avoided by disabling the animation effect on the Nav Bar.

Navigation Stack

You can use the $pop property to "pop" or remove a specified number of items off the navigation stack: it is analogous to clicking on the left or back button, since it allows you to step back in the navigation stack a specified number of times.

The $pop property can only be assigned at runtime. If you try to pop more items off the stack than exist, it will pop everything except the first item. If you assign to $pop, note that the evUserChangedPage event of a linked paged pane will not be triggered.

In addition, you can set the text or title for the left (back) button for a navbar. If provided, the 6th col in the row assigned to $push allows you to specify the left button text. This can be set to an empty string to default to the previous page's title.

Events

The Nav Bar reports evClickInitialLeftButton when the initial left button has been clicked, and evClickRightButton when the right button has been clicked.

The events evPushFinished and evPopFinished are triggered when the push or pop animations complete. Both events have one event parameter which is the associated page number that has been pushed or popped.

The evWillPop event is sent before an item is popped from the navigation bar stack, for example, when the user clicks on a left button. It has one parameter, pPageNumber, which is the number of the page that will be popped. It is a client-only event. For example, this event can be used to prevent the pop from occurring by discarding the event with:

```
Quit event handler (Discard event)
```

Example

The following Navigation Bar has a main title and a button on the right which is used to display some information on the second pane of a paged pane.

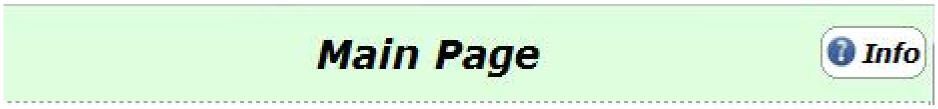


Figure 190:

The Nav Bar is placed across the top of the form and its various properties under the General and Appearance tabs in the Property Manager are set, as follows:

Property	Description
$events	set to receive evClickRightButton events
$linkedobject	set to pPane, the name of the paged pane
$push	can only be assigned at runtime; see below
$initialtitle	set to “Main Page”
$initialrighticonid	set to 1794, the id of an icon in Omnispic
$initialrighttext	set to “Info”

The $event method for the Nav Bar ($name = oNav) traps a user click on the right button, and has the following event code:

```
On evClickRightButton
  Do $cinst.$doPush(2,'Info','','','')
```

The $doPush method is a class method in the remote form and has the following parameters, variables, and code.

```
# PaneNo (Short Int)
# Title, RightBtnText, RightBtnIcon (must be an icon url), NoLeftBtn (all Character)
# lRow is local var of type Row
Do lRow.$define('New class','Title','text for right btn','icon url for right btn','no left btn',’text for left
Do lRow.$assigncols(PaneNo,Title,RightBtnText,RightBtnIcon,NoLeftBtn,LeftBtnText)
Do $cinst.$objs.oNav.$push.$assign(lRow)
```

The effect of assigning to the $push property is to change the pane number in the paged pane specified in the $linkedobject property which, in this case, displays some information for the end user on the second pane. Assigning to $push adds a left or ‘Back’ button to the navbar which the end user can use to go back to the previous pane; the default text for this button is the name of the previous pane, or you can pass your own text in the 6th column of the row passed to $push.

Navigation Menu Object

Group	Icon	Name	Description
Navigation		Navigation Menu	Dropdown menu with hierarchical

The **Navigation Menu Object** (or Nav Menu) allows you to build interactive cascading menus within your remote forms, providing a navigation method similar to that found on many websites. Such menus typically comprise a number of hot text links, which cause further menus to open below the top level. In addition, each menu option can have an image as a background or link. The nav menu can also operate as a breadcrumb, with a hierarchical set of text links, similar to the folder selection mechanism found in Windows Explorer.

The Nav Menu control has been implemented for both the JavaScript client (remote form class) and the fat client (window class), and the two controls have almost identical properties, with the same look and feel. Unless otherwise stated, the descriptions here apply to both the JavaScript Client and fat client types of control.

There is an example app called **JS Navigation Menu** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

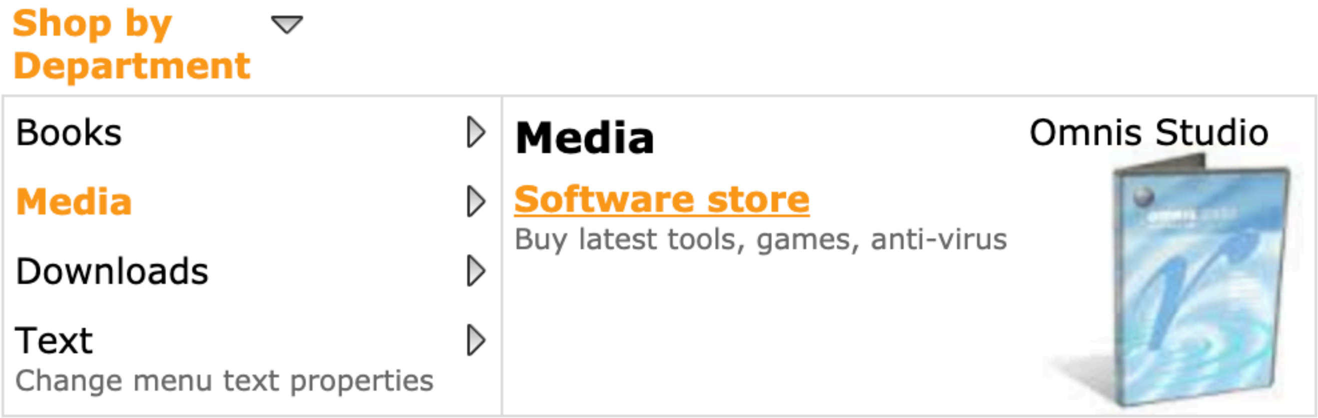


Figure 191:

Menu List Content

The Navigation Menu control uses a list specified by $dataname to define its content. The list should have *seven columns* with each line of the list corresponding to a single entry in the menu. The columns in the list are defined as follows:

Column	Type	Description
Type	Integer	The menu entry type, a kNavMenuType… constant. See below.
Text	Char	The text for the menu entry. This can include styles embedded using the style() function, and embedded kCr characters in order to split the text over multiple lines.
Desc	Char	Optional text describing the menu entry. This can include styles embedded using the style() function, and embedded kCr characters in order to split the description over multiple lines.
Flags	Integer	Sum of one or more constants that indicate how the menu entry behaves. See table below.
Ident	Integer	A unique integer value that identifies the menu entry. Note that the control does not enforce uniqueness.
Tag	Char	A unique string value that identifies the menu entry. Note that the control does not enforce uniqueness. In fact, developers may choose to just use the tag or just use the ident, or make the combination of tag and ident unique.

Column	Type	Description
Info	Row	A row that contains further information required for the menu entry. Not used for all entry types. E.g. contains the content for a cascaded menu, see kNavMenuTypeCascade.

The flags column can be zero, or a sum of one or more of the following flag values and specifies how the menu entry behaves:

Flag	Description
kNavMenuFlagHorizontalLayout	Only applies to the first line of a menu (or cascaded menu) list. If not set, entries are laid out vertically; if set, entries are laid out horizontally.
kNavMenuFlagEndOfRowOrColumn	If set, the menu entry for this line is the last entry in the current row or column in the layout for the menu (row or column depends on whether layout is horizontal or vertical respectively)
kNavMenuFlagDisabled	If set, the menu entry is disabled. This means it will not accept clicks, and it will not hot-track.
kNavMenuFlagBreadcrumb	Only applies to line 1 of the $dataname menu list. If set, the menu is a breadcrumb control, and for the $dataname list, kNavMenuFlagHorizontalLayout is turned on and kNavMenuFlagEndOfRowOrColumn is ignored.
kNavMenuFlagBreadcrumbSeparator	If set and kNavMenuFlagBreadcrumb is set and applies to line 1, the entry draws the breadcrumb separator. Note that the control uses the description text color ($descriptiontextcolor) as the color of the separator.

Menu Types

The first column of your data list sets the type of menu control, using one of the following kNavMenuType… constants:

kNavMenuTypeHeading

Used as a heading to group other menu entries. Typically, this would be a disabled entry, but it can accept clicks if desired. The info column is not used for this type.

kNavMenuTypeEntry

A normal menu entry, typically used to accept a click and generate an event.

kNavMenuTypeImage

A menu (or cascaded menu) list can only have a single image entry (others are ignored). You use the info column to provide an image that will be displayed as a background of the menu, and which will also accept clicks. The control will place the image at the “end” of the menu, irrespective of where the entry is placed in the list. The info column for an image entry is a row with the following columns. Note that only the icon column is mandatory.

Column	Type	Description
Icon	Character or Integer	For the JavaScript Client, the character URL of the image, generated by calling iconurl(iconid). For the fat client, the integer icon id of the image.
Horizontal offset	Integer	You can adjust the horizontal position of the image in the menu by supplying a value here. Defaults to zero.
Vertical offset	Integer	You can adjust the vertical position of the image in the menu by supplying a value here. Defaults to zero.

kNavMenuTypeCascade

An entry representing another menu which cascades from the entry when either the mouse is over the entry, or when the entry is clicked (this depends on the $openwhenmouseover property described below). Note that the control supports a cascade nesting depth of 15.

The info column for a cascaded menu entry is a row with the following columns (note: when you understand how the properties and events for the control work, you will see that the info row does not always need to be fully specified for a cascaded menu - in many cases only column 1 is required, and in fact, in some cases, the info row is not required at all for a cascaded menu):

Column	Type	Description
List	List	A menu list defining the entries in the cascaded menu.
Cascade flags	Integer	Zero, or a sum of kNavMenuCascadeFlag... flags, see below. Defaults to $defaultcascadeflags.
Open side	Integer	The side from which the cascaded menu will open. Either kNavMenuSideLeft, kMenuSideRight, kMenuSideBottom or kMenuSideTop. Defaults to $defaultcascadeopenside.
Border edges	Integer	A sum of kNavMenuSide... constants that specifies the edges of the cascaded menu that are to have a border. Defaults to $defaultcascadeborderedges.
Border color	Integer	The color of the border of the cascaded menu. Defaults to $defaultcascadebordercolor. For the JavaScript Client control, you must set this column to the result of truergb(color) if the color you are using is a color constant.
Border width	Integer	The width of the border of the cascaded menu (1-16). Defaults to $defaultcascadeborderwidth.
Background color	Integer	The background color of the cascaded menu. Defaults to $defaultcascadebackcolor. For the JavaScript Client control, you must set this column to the result of truergb(color) if the color you are using is a color constant.
Background alpha or foreground color	Integer	For the JavaScript Client, the background alpha value for the cascaded menu (0-255). Defaults to $defaultcascadebackalpha. For the fat client, the foreground color of the cascaded menu. Defaults to $defaultcascadeforecolor.

Column	Type	Description
Background pattern	Integer	Only applies to the fat client. The background pattern of the cascaded menu. One of the standard pattern constants. Defaults to $defaultcascadebackpattern.

Cascade Flags

The cascade flags are as follows:

Flag	Description
kNavMenuCascadeFlagUseEventToPopulate	If set, the control sends evLoadCascade in order to populate the cascaded menu
kNavMenuCascadeFlagUseEventWhenRequired	If set, and kNavMenuCascadeFlagUseEventToPopulate is also set, only send evLoadCascade when data is required again for some reason,rather than each time the menu opens
kNavMenuCascadeFlagOpenOnParentEdge	If set,the cascaded menu opens on the relevant edge of the parent menu, rather than opening on the relevant edge of the parent entry.
kNavMenuCascadeFlagExpand	If set, and kNavMenuCascadeFlagOpenOnParentEdge is also set, the cascaded menu expands if necessary to the width or height of the parent.
kNavMenuCascadeFlagUseDefault	If set, use the default cascaded menu flags for the control ($defaultcascadeflags) and ignore any other flags. This allows you to use default cascade flags, and then override other properties using the info row.

Menu Properties

In addition to the standard control properties, the Nav Menu control has the following properties:

Property	Description
$borderedges	A sum of kNavMenuSide… constants that specifies the edges of the control that are to have a border.
$borderwidth	The width of the border of the control (1-16).
$closeboxiconid	The icon id of the close box for cascaded menus (only relevant when $openwhenmouseover is kFalse, and the control is not in breadcrumb mode). Note that when $openwhenmouseover is kFalse, on a non-touch device, menus still close automatically when the mouse leaves the control or its open cascaded menus. If you do not want a close box, set this to zero. On a touch device you can close all open cascaded menus by touching an area away from the control and its open cascaded menus.

Property	Description
$defaultcascade…	Default properties for cascaded menus (see the description of the info row for cascaded menus). These eliminate the need to repeat this information for every cascaded menu.
$horizontalcascadeiconid	The id of the icon used to represent an entry that cascades to the left or right.
$horizontalspacing	Horizontal spacing used when laying out entries.
$hotcloseboxiconid	The icon id of the close box for cascaded menus, used when $openwhenmouseover is kFalse, and the mouse is over the close box.
$verticalcascadeiconid	The id of the icon used to represent an entry that cascades to the top or bottom.
$verticalspacing	Vertical spacing used when laying out entries.
$font… properties	Used to control the font and colour of entry text: $font, $fontsize, $fontstyle, $textcolor, $hotfontstyle, $hottextcolor
$descriptionfont… properties	Used to control the font and colour of description text: $descriptionfont, $descriptionfontsize, $descriptionfontstyle, $descriptiontextcolor
$headingfont… properties	Used to control the font and colour of heading text: $headingfont, $headingfontsize, $headingfontstyle, $headingtextcolor, $hotheadingfontstyle, $hotheadingtextcolor
$openwhenmouseover	If true, cascaded menus open when the mouse is over the relevant part of the control. Otherwise, the user needs to click in order to open a cascaded menu. On a mobile device, the value of this property is ignored and treated as kFalse, because there is no mouse.

Menu Events

When an entry is selected in the Nav Menu an event is triggered, one of the the following events:

Event	Description
evLoadCascade	The control sends this event when it needs to populate a cascaded menu (i.e. kNavMenuCascadeFlagUseEventToPopulate is set for the menu). The application code processing this event builds a list for the cascaded menu, and assigns it to the runtime-only property $cascadecontents.
evMenuEntryClicked	The control sends this event when the user clicks on a menu entry.
evEmptySpaceClicked	The user has clicked in empty space to the right or bottom of the menu (generated for top-level menu page only).

evLoadCascade and evMenuEntryClicked have 2 event parameters, pLineIdent and pLineTag, which are the ident and tag of the menu list line for which the event was generated. These events and their parameters can be trapped in the $event method for the control.

Scrolling

If the initial menu (set with the list content in $dataname) is too wide to fit the control, the control uses scroll arrows at the left and right to allow its content to be scrolled – ideally you should fit your content to the width of the control, so no scroll arrows are required. The scroll arrows are displayed in this case to support the breadcrumb mode which uses a single row of entries for the initial page.

On a touch device, the scroll arrows are not displayed but you can scroll the control horizontally by touch-dragging. The scroll arrows are the same ones used for the tab control which are specified in core.css (img.ctrl-arrow-left-dis & img.ctrl-arrow-right-dis) and the images folder, so you can change their appearance if required. Note that the core.css file is minified, so you may want to use a prettifier tool to make it more human-legible. Do not edit the classes in core.css, rather you should override the classes by adding your own version in the user.css file found in the html\css folder in your Omnis development tree.

Navigation Menu Example

The following methods define the Nav Menu for a fictional online shop (they are contained in an object class and called via an object variable in the remote form). An example containing a similar Nav menu is available in the JavaScript Components Gallery on the Omnis website, and in the Hub.

First the content list for the Nav Menu is defined (with seven columns), then the second-level items are created, in this case the shop departments, and the top-level for the menu is added. The list built here is added to the list specified in $dataname for the Nav Menu object in a remote form.

```
Do method defineMenuList (lDepartmentList)
Do lDepartmentList.$add(kNavMenuTypeCascade, "Books", "", 0, 100, "books")
Do lDepartmentList.$add(kNavMenuTypeCascade, "CDs", "", 0, 101, "cds")
Do lDepartmentList.$add(kNavMenuTypeCascade, "Digital music", "", 0, 102, "digimusic")
Do method defineMenuList (lOmniShop)
Do lOmniShop.$add(kNavMenuTypeCascade, con("Shop by",kCr,"Department"),"",0,1,"", row(lDepartmentList,kNavMenu
Quit method lOmniShop
```

The defineMenuList method is a generic method to define the list of the menu content:

```
Do pMenuList.$define(lType,lText,lDescText,lFlags,lIdent,lTag,lInfo)
```

The $events property for the Nav Menu object has two events specified: evLoadCascade and evMenuEntryClicked. The menu object itself has the following event method:

```
On evLoadCascade
  Do iNavMenuObject.$shopLoadCascade($cfield,pLineTag)
On evMenuEntryClicked
  If not(iNavMenuObject.$handleclick($cfield,pLineTag))
End If
```

The $shopLoadCascade method builds the content for the cascaded menu:

```
Do method defineMenuList (lCascadeList)
Switch pTag
  Case "books"
    Do lCascadeList.$add(kNavMenuTypeHeading,"Books","",kNavMenuFlagDisabled)
    Do lCascadeList.$add(kNavMenuTypeEntry,"Best sellers","Top 1000 books",0,200)
    Do lCascadeList.$add(kNavMenuTypeEntry,"eBooks","For kindle and tablets",0,201)
    Do lCascadeList.$add(kNavMenuTypeImage,"Pre-order","",0,202,"",row(iIcon10001))
    # this line is shown in the pic below
  Case "cds"
    Do lCascadeList.$add(kNavMenuTypeHeading,"CDs","",kNavMenuFlagDisabled)
    Do lCascadeList.$add(kNavMenuTypeEntry,"CD store","Over 3m CDs",0,300)
    Do lCascadeList.$add(kNavMenuTypeImage,"Rock store","",0,301,"",row(iIcon10002))
  Case "digimusic"
```

```
    Do lCascadeList.$add(kNavMenuTypeHeading,"Digital music","",kNavMenuFlagDisabled)
    Do lCascadeList.$add(kNavMenuTypeEntry,"Music store","Over 30m songs",0,400)
    Do lCascadeList.$add(kNavMenuTypeEntry,"Your music library","Play online",0,401)
    Do lCascadeList.$add(kNavMenuTypeImage,"Pre-order","this new album",0,402,"",row(iIcon10000,-200,-50))
  End Switch
Calculate pControl.$cascadecontents as lCascadeList
```

The menu will look something like this:

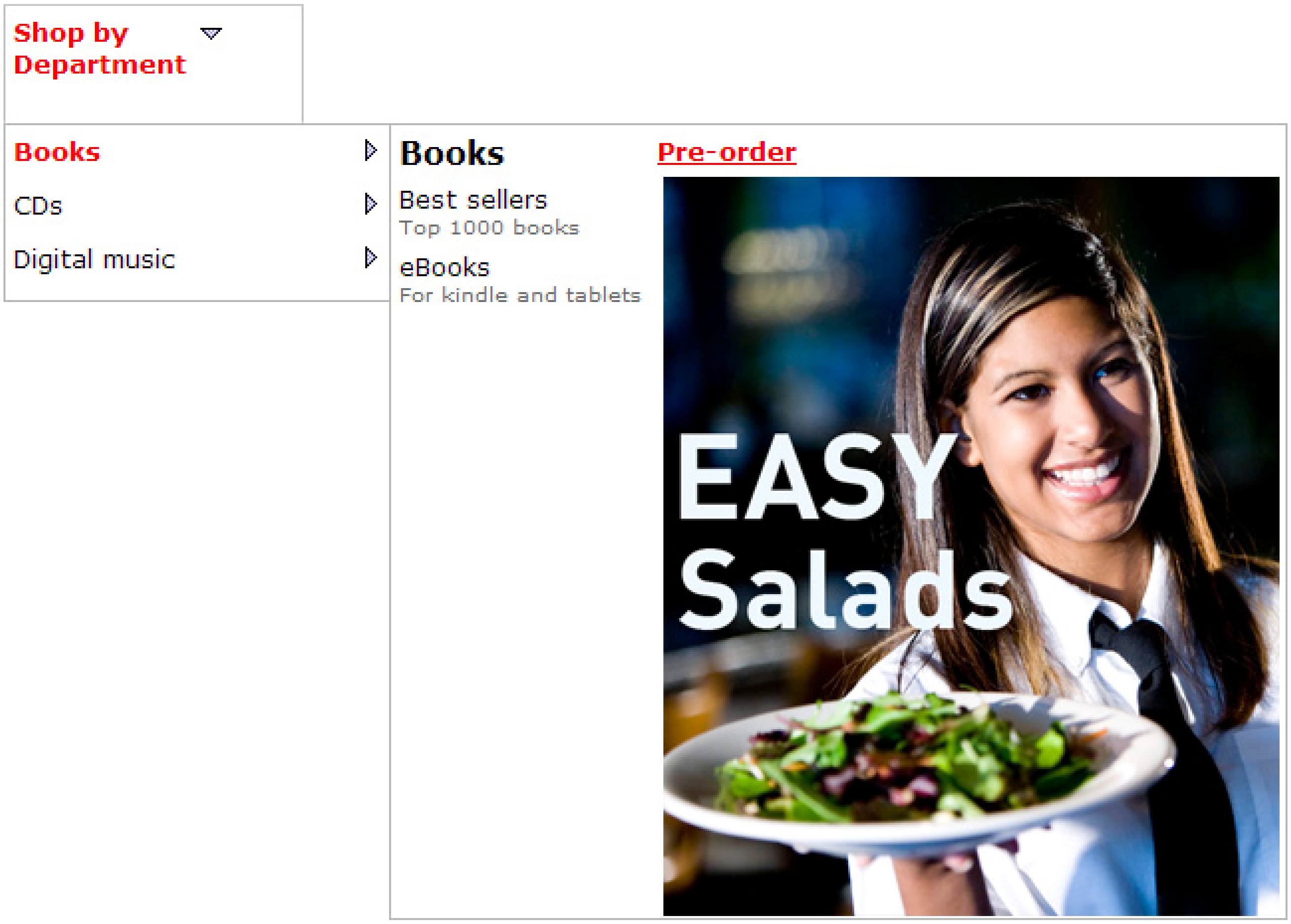


Figure 192:

Page Selector

Group	Icon	Name	Description
Navigation	♦♦♦	Page Selector	Allows selection of page pane usin

The **Page Selector** (or Page control) links to a Paged pane on a remote form and allows the end user to change the current page in the linked paged pane by swiping over the Page selector or clicking for non-touch screens. The Page selector also gives the end user a visual clue as to the current selected pane in the linked page pane object, since the highlighted dot in the control changes to reflect the current page in the linked paged pane.

There is an example app called **JS Page control** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

Figure 193:

The paged pane linked to the Page selector is specified in the $linkedobject property. In this case, when the page control is clicked or swiped the linked paged pane control will select the next available pane automatically.

Property	Description
$::currentpage	the current page number
$linkedobject	the name of a paged pane object on the current remote form that links to the iPage cont
$::pagecount	the number of pages
$pageindicatorcolor	the color for the current page indicator
$currentpageindicatorcolor	the color for all the pages except the current page indicator

When the page indicator changes in the Page selector an evPageChanged is triggered containing the number of the new page in pValue.

Paged Pane

Group	Icon	Name	Description
Containers	❐	Paged Pane	Can contain fields & other objects o multiple panes

The **Paged Pane** provides a very convenient method to show a number of fields or controls *grouped together* on separate panes, or to break down an entry form into more manageable parts whereby each pane contains a small number of fields. The **JS Page Control** example app in the **Samples** section in the **Hub** in the Studio Browser uses the Paged Pane; the same app is available in the JavaScript Component Gallery.

The $pagecount property specifies the number of panes, and $currentpage specifies the current pane. In design mode, you have to set $currentpage to the number of the pane you wish to add fields to, or you can right-click the background of the paged pane and select the number of the pane you want to edit. You can set $effect to select different border effects for the control (a kJSborder… constant).

You can link a paged pane to a Navigation Bar, Page Selector, or Tab Bar control so when the nav bar, page or tab changes the current pane of the paged pane changes accordingly. To link a paged pane to one of these controls, set the $linkedobject property of the Nav bar, Page Selector or Tab bar to the name of the paged pane.

By setting $scrolltochangepage to kTrue the pages are laid out horizontally, and the end user can change the current page by scrolling horizontally (for touch devices the end user can change panes by tapping on the current pane).

Events

An evUserChangedPage event is triggered when the pane changes and the new page number reported in pPageNumber.

When a Paged Pane is linked to a Nav bar or Tab bar control, the evUserChangedPage event is triggered when the nav button or tab is clicked to change the pane.

Rounded Corners

The $borderradius property lets you add rounded corners to the Paged pane. The radius can be specified by a single value, so all corners are the same radius, or up to four hyphen-separated pixel values, in the order topleft, topright, bottomright, bottomleft, e.g. 4-4-0-0 to add rounded corners to just the top of the paged pane. If the bottomleft value is omitted the topright value is used. If bottomright is omitted the topleft value is used. If topright is omitted the topleft value is used.

Note: If a border radius is set, the rounded corners are not drawn in design mode: they are only rendered when the app is run on the client. The rounded corners are not drawn in design mode to allow the full use of the available space within the page pane control while designing the form.

Using $dataname

The Page Pane control has a $dataname property which you can use to set the value of $currentpage. When the form is opened or redrawn the numeric value of $dataname is used to set the current page. If $dataname is empty or returns an invalid page number, the control uses the page number in $currentpage.

Page Panes in Complex grids

You can use a page pane in a complex grid. You can set the value of $currentpage by assigning a column in the complex grid list to $dataname of the page pane. Therefore each row in the complex grid could display a different page in the page pane control.

In this case, controls within the page pane control will also get their data from the complex grid control, if their $dataname refers to a column in the list used to build the complex grid.

Group Boxes

There is no Group box JS component, but you can create one “on the fly” using the $makegroupbox() method to change a Paged pane into something that simulates the behavior and appearance of a group box.

The method PagePaneName.$makegroupbox(cLabel[,cFont,cFontSize,cTextColor]) converts a Paged pane into a Group box with the specified label, as well as the optional CSS font, font size, and text color. The method must be executed on the client, and can be called from $init in the remote form.

Animated Transitions

If enabled, the $animatetransitions property ensures that the transition between pages is animated when the current page is changed. The property cannot be changed at runtime (the same as $scrolltochangepage).

If used in conjunction with $scrolltochangepage, when the user stops scrolling, the pane will smoothly animate into position, rather than jumping instantly.

The animation time is set to 500ms, which should be fine for most purposes, but if you wish to change this, you can use JavaScript to change the Paged Pane control’s (or its prototype’s) ANIMATION_TIME property.

Page Styling

Page panes have a default CSS classname ‘omnis-pagedpane-page’. This allows you to apply CSS styling or behavior to each page of the paged pane control.

In addition, a CSS rule (-webkit-overflow-scrolling: touch;) enables momentum scrolling on iOS, i.e. for touch iOS devices, scrolling slows down before stopping.

Picture Control

Group	Icon	Name	Description
Media	🖼	Picture	Standard field for displaying image

The **Picture Control** allows you to display an image in your form: you can display an image file in a folder, an image from a database, or an icon (including an SVG icon), depending on the combination of settings of $dataname, $mediatype and $iconid. If $mediatype is empty (and $iconid is zero), the $dataname of the Picture control is a URL to the image to be displayed, relative to your html page containing your remote form (i.e. the JavaScript Client). If $mediatype is specified, then $dataname is the name of a binary instance variable containing the image data: in this case, $mediatype can be set to one of the standard image types, e.g. image/png, image/jpeg or image/gif. Alternatively, $iconid can be set to a URL referencing an image file in the 'html/icons' folder, overriding the $dataname and $mediatype properties. For backwards compatibility, the picture control can display an icon in an icon data file (Omnispic) or #ICONS by setting $iconid to a numeric icon ID.

The $tintcolor property allows you to apply a tint color to themed SVG icons, that is, any SVG icons that have been themed using the JS Themer tool (available in the **Tools>>Add ons** menu). $tintcolor is ignored for all other image formats.

There is an example app called **JS Picture** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

Image Alignment and Scaling

The $picturealign property specifies where the picture will be positioned in the control and is a kPAL... constant. The $noscale property determines whether or not the images displayed in the control are scaled. The $keepaspectratio property determines whether or not the images displayed in the control keep their aspect ratio when scaled: if true, and $noscale is false, the aspect ratio of pictures is maintained when they are scaled.

The property $keepaspectratiomode controls how the image in the Picture control is scaled and positioned when $keepaspectratio is true and $noscale is false. The value of $keepaspectratiomode is a kKAR... constant with the possible values:

- **kKARtopLeft**
  The image is scaled to fit the control and anchored at the top-left corner. This is the default value (and maintains compatibility with existing libraries)
- **kKARcenter**
  The image is scaled to fit the control and centered, so background may be visible at the top and bottom or the left and right of the image, depending on the shape of the image control and the orientation of the image
- **kKARfill**
  The image is scaled to fill the control and centered, so no margin (background) is shown. This can result in the image being cropped at the top and bottom, or the left and right, depending on the shape of the image control and the orientation of the image

Example

In the **Webshop** sample app, the product images are shown in a Picture control embedded in the Complex grid control on the main jsShop remote form. In this case, $mediatype is set to JPG and the $dataname of the control is iProductList.product_picture which holds the image data for each product.

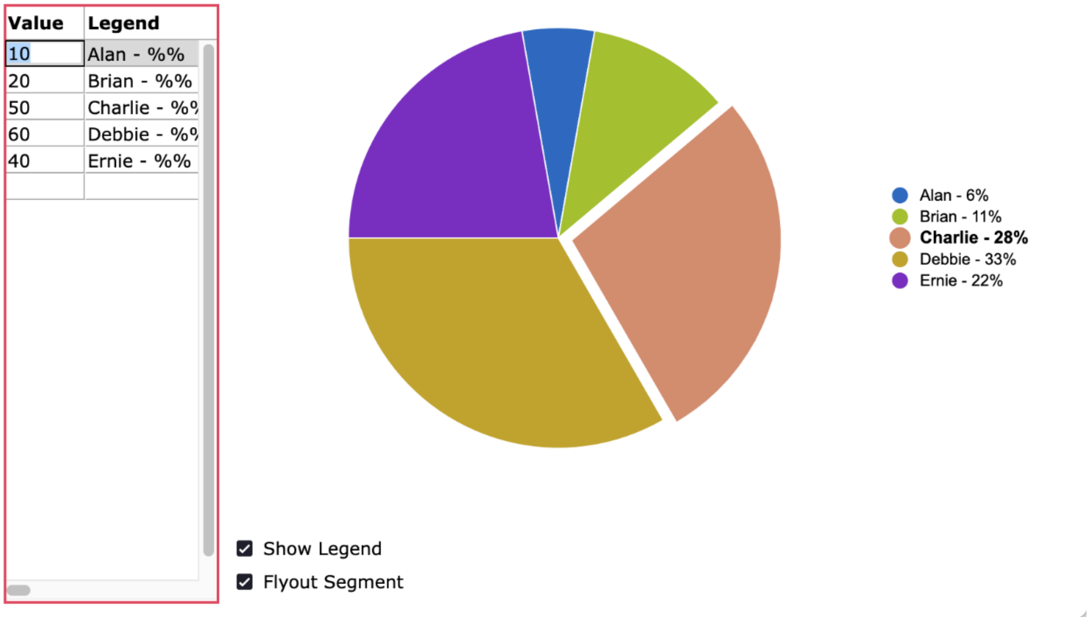
Pie Chart Control

Group	Icon	Name	Description
Visualization	◔	Pie Chart	Displays a pie chart based on a list

The **Pie Chart** control allows you to display a simple dataset contained in a list variable as a pie chart. See the description for the Bar Chart Control for information about defining the list variable structure for pie and bar charts, plus chart events, as well as information about setting the bar/segment colors for charts, including the use of theme colors and the $colorlist runtime property to set the segment colors.

Each value in a **Pie chart** is shown as a segment, and its angle indicates a *percentage of the total* of all the values in your list, that is, the angle in degrees is calculated as the proportion of the *individual value* when compared to the *total of all values* in the list.

There is an example app **JS Pie Chart** in the **Samples** section in the **Hub** in the Studio Browser showing you how to set up a Pie chart using a simple list of data, and the same app is available in the JavaScript Component Gallery.



Value	Legend
10	Alan - %%
20	Brian - %%
50	Charlie - %%
60	Debbie - %%
40	Ernie - %%

☑ Show Legend

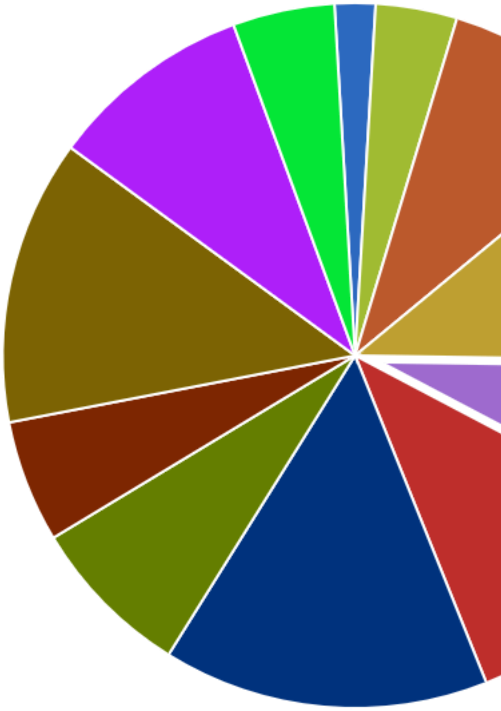
☑ Flyout Segment

Figure 194:

Main and Legend Titles

There are a number of properties in the Pie Chart component to allow you to add a legend title and have some control over the appearance and positioning of the legend.

Property	Description
$maintitle	The legend title
$legendnames	If true the legend shows the value names (column 2) and not values (column 1). The list data structure is same as bar chart
$legendcolumns	The number of columns the legend is split into
$legendpos	Changes the position of the legend: can be above, below, left or right of the pie, or use kJSPieLegendOff to hide the legend
$flyout	Enables the pie segments to move or “fly out” when the end user’s pointer hovers over the segment (kFalse by default)



The following pie chart has $legendnames = kTrue, $legendcolumns = 2 and $flyout = kTrue.

Two positioning constants are available for the $legendpos property (which in practice are only appropriate for mobile devices), whereby as the device is rotated and the screen orientation changes, the legend is repositioned automatically either “before” (left or above) the pie or “after” (right or below) the pie. The constant values are:

- **kJSPieLegendAutoBefore**
  the legend is placed before the pie chart, either above or to the left of the chart.
- **kJSPieLegendAutoAfter**
  the legend is placed after the pie chart, either below or to the right of the chart.

Popup Menu Control

Group	Icon	Name	Description
Menus	☰	Popup Menu	A menu that pops up when clicked

The **Popup Menu Control** is a menu that pops up when the user clicks on the header of the control, or when $hotwhenmouseover is true the menu will pop up when the end user’s pointer hovers over the control. The contents of the popup menu can be:

- a *list variable* specified in $::listname
- a *remote menu class* specified in the $::menuname property, or

When specifying one of these properties, the other property must be empty; the properties are on the Data tab in the Property Manager.

There are two example apps that use the Popup menu in the **Samples** section in the **Hub** in the Studio Browser: the first is named **JS Droplist, Combo, Popup** and uses a popup built from the list; the second uses a Remote menu to display a popup menu. The same apps are available in the JavaScript Component Gallery; the following screen shows the ‘red night’ JS Theme in use.

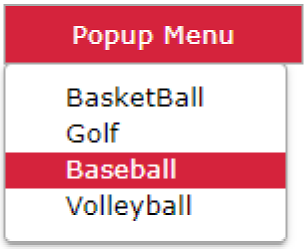


Figure 195:

When using a list variable to populate a Popup menu, the data in the column specified in $coltext is used for the menu options. The $colenabled property is the column name for the menu line enabled state, and $colcommandid is the column name holding the menu line command id.

You can add an icon to the popup menu by setting $iconid to the ID of an icon in an icon file or #ICONS. You can place the icon before or after the menu title by setting $textbeforeicon.

The menu will normally popup when the user clicks on the control, but you can make the menu popup when the end user's pointer passes over the control (it is "hot") by setting $hotwhenmouseover to kTrue.

You can control the position of the popup by setting $menupos to one of the constants: kJSPopMenuPos**Bottom,** kJSPopMenuPos**Right,** or kJSPopMenuPos**Top.**

When the menu is clicked the evClick event is triggered with the selected line reported in pLinenumber. You can use the following $event method to trap the line number:

```
On evClick
  If pLineNumber>0   ## a line was selected
    ## Do something
  End If
```

You can use an HTML <select> tag for the user interface by setting $usehtmlselect to kTrue.

Menu Line Height

The $menulineheight remote form property allows you to set the line height *for all the menus in a remote form,* including Popup menus and any Context menus for the form, as well as menus belonging to Tab strip and Splitbutton controls.

For pre-Studio 10.2 converted applications, the value of $menulineheight will be zero, meaning the font size will determine the line height, as previous versions. For new applications, this will be a touch-friendly value to give enough space for each menu option.

Progress Bar Control

Group	Icon	Name	Description
Other		Progress Bar	Shows progress of server process c calculation

The **Progress Bar Control** lets you display a progress bar in your remote form, to indicate the progress of some process in your application; this could be a static value, or a value that changes dynamically. There is an example app called **JS Progress** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

The current value of the progress bar is reported in the $::value property which is a value between 0 and the value of $max inclusive. The color of the bar representing the completed amount can be set in $progresscolor, which only applies when the standard HTML5 progress control is not available. The following screen is when $usesystemappearance is kTrue on Windows:

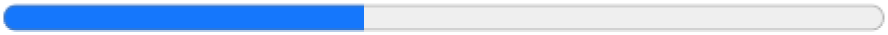


Figure 196:

System Appearance

When set to true (the default), the $usesystemappearance forces the progress control to use the <progress> HTML5 element (if it's supported by the browser). If $usesystemappearance is set to false, the progress control uses two <div> elements, and the following additional properties apply: $secondarycolor sets the color of the stripes of the progress bar, and $progressanimation (true by default) animates the progress bar. The following progress bar has $usesystemappearance set to false (and using the 'red night' JS Theme):

Figure 197:

Example

The following example code (from the example app in the Hub) assumes the progress control has been added to a remote form and a button is used to initiate some process and send a *carryon* event to the progress itself; note the 'lockui' client command is used to stop any clicks on the UI once the progress is initiated (except for the Cancel button). The initial values for $::value and $::max of the progress control are 0 and 100 respectively. The following code could be behind a button:

```
On evClick
  Do $cinst.$clientcommand("lockui",row(kTrue))
  Calculate iCancelled as kFalse
  Calculate iValue as 1
  Do $cinst.$objs.ProgressBar.$sendcarryon.$assign(kTrue)
```

The evCarryOn event is sent to the progress bar which has the following event method:

```
On evCarryOn
  If not(iCancelled)
    Calculate iValue as iValue+5
    Do $cinst.$objs.ProgressBar.$::value.$assign(iValue)
    If iValue<100
      Do $cinst.$objs.ProgressBar.$sendcarryon.$assign(kTrue)
    Else
      Do $cinst.$clientcommand("lockui",row()) ## unlock ui
    End If
  End If
```

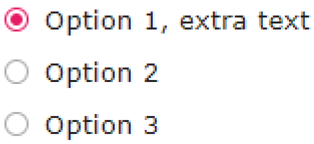
Radio Button Group

Group	Icon	Name	Description
Buttons	● Opt1 ● Opt2	Radio Button Group	Displays a group of radio buttons f exclusive selection

The **Radio Button Group** control presents a number of mutually exclusive Radio buttons that can be either on or off: selecting one of the radio buttons *deselects all other buttons* in that group. The variable you assign to a radio group should be numeric (e.g. Integer). Its value is within the range $minvalue and $maxvalue inclusive and directly corresponds to which button in the group is selected, that is, the first button selects the first value in the range, the second button the second value, and so on. The Radio button group has the following properties:

Property	Description
$dataname	a numeric variable
$::horizontal	If true, the radio column order is horizontal
$columncount	The number of columns shown for the radio group
$minvalue	The minimum value for the radio group
$maxvalue	The maximum value for the radio group
$text	Comma-separated list of labels assigned to the buttons, or double comma to include a comma in any lab
$radiobuttoncolor	The color for the Radio group control

The labels for the buttons are assigned in $text which is a comma-separated list (e.g. All,Female,Male). You can include a comma in the text for any label by adding a second comma. For example, when $text is set to: Option 1,, extra text,Option 2,Option 3 (and $::horizontal = kFalse, $colcount = 1), the following radio button group is displayed:



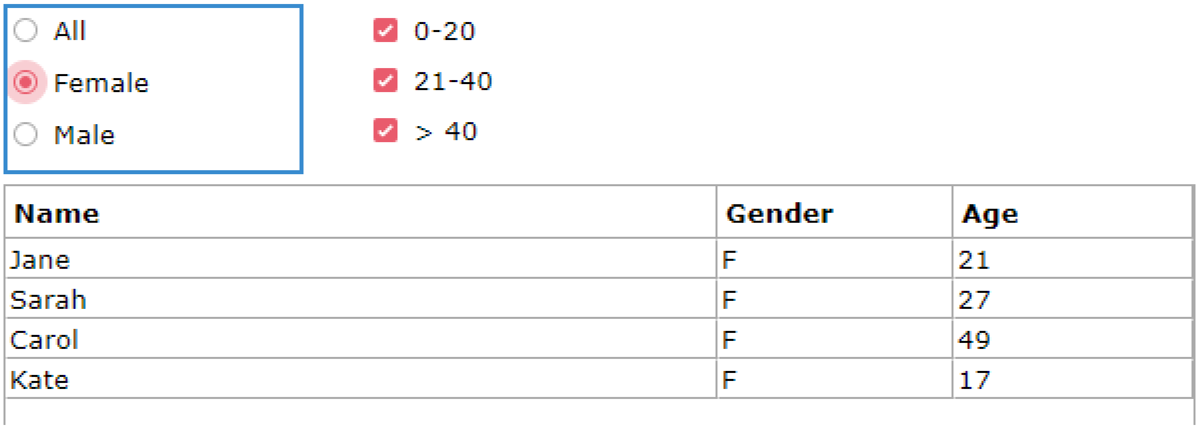
● Option 1, extra text
○ Option 2
○ Option 3

Figure 198:

Events

When a button in the Radio button group is clicked the evClick event is reported with pNewValue containing the value of the selected button.

The **JS Radio and Checkbox** example app in the **Samples** section in the **Hub** in the Studio Browser uses a Radio button group to select Gender, and the same app is available in the JavaScript Component Gallery; the following screen shows the 'health' JS Theme in use.



○ All
● Female
○ Male

☑ 0-20
☑ 21-40
☑ > 40

Name	Gender	Age
Jane	F	21
Sarah	F	27
Carol	F	49
Kate	F	17

Figure 199:

Example

The **Webshop** app uses a Radio button group control to allow the end user to select a group or category of products to be shown in the main product list. (To examine this control and its properties and methods, open the webshop library and the jsShop remote form.) The $minvalue and $maxvalue of the Radio button group are set to 0 and 8, respectively (although the groups are generated dynamically in the form), and the numeric variable iRadioGroup with an initial value of 1 is assigned to $dataname. When the form is opened, the $construct() method behind the Radio button group calls a $build method.

```
# radiogroup control is called 'filter' containing
# $construct method which is run when the form opens
```

```
Do iGroupList.$definefromsqlclass($tables.T_qGroups)
Do $cfield.$build()
```

The $build method generates a list of product groups from the database, which is then concatenated into a single comma-separated list and assigned to the $text property of the Radio button group.

```
# $build method behind the radiogroup
Do iGroupList.$selectdistinct()
Do iGroupList.$fetch(kFetchAll)
For iGroupList.$line from 1 to iGroupList.$linecount() step 1
  # loop through the list
  Calculate text as con(text,mid($prefs.$separators,3,1),iGroupList.product_group)
  # uses the localized separator (possibly semicolon)
End For
Calculate text as mid(text,2,len(text))
Do $cfield.$minvalue.$assign(1)
Do $cfield.$maxvalue.$assign(iGroupList.$linecount())
# $maxvalue of radiogroup is set to the number of groups in list
Do $cfield.$text.$assign(text)
```

When the form is opened the main product list is built using a method behind the product list itself (also called $build) and the list initially contains the Appetizers only. A group of radio buttons is created, each item representing a different group or category of food or drink; the initial value of the Radio button group is set to 1 selecting the first item in the group.

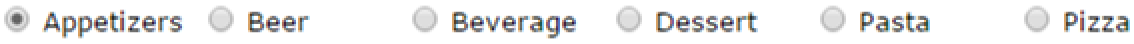


Figure 200:

When the end user clicks on the Radio button group, to select another product type, the click is detected in the $event method in the radio button group control, the number of the radio button clicked is passed in pNewVal, and the product list is rebuilt based on the selected product group; note a Where clause is created based on the selected group and sent to the $build method behind the main productList control.

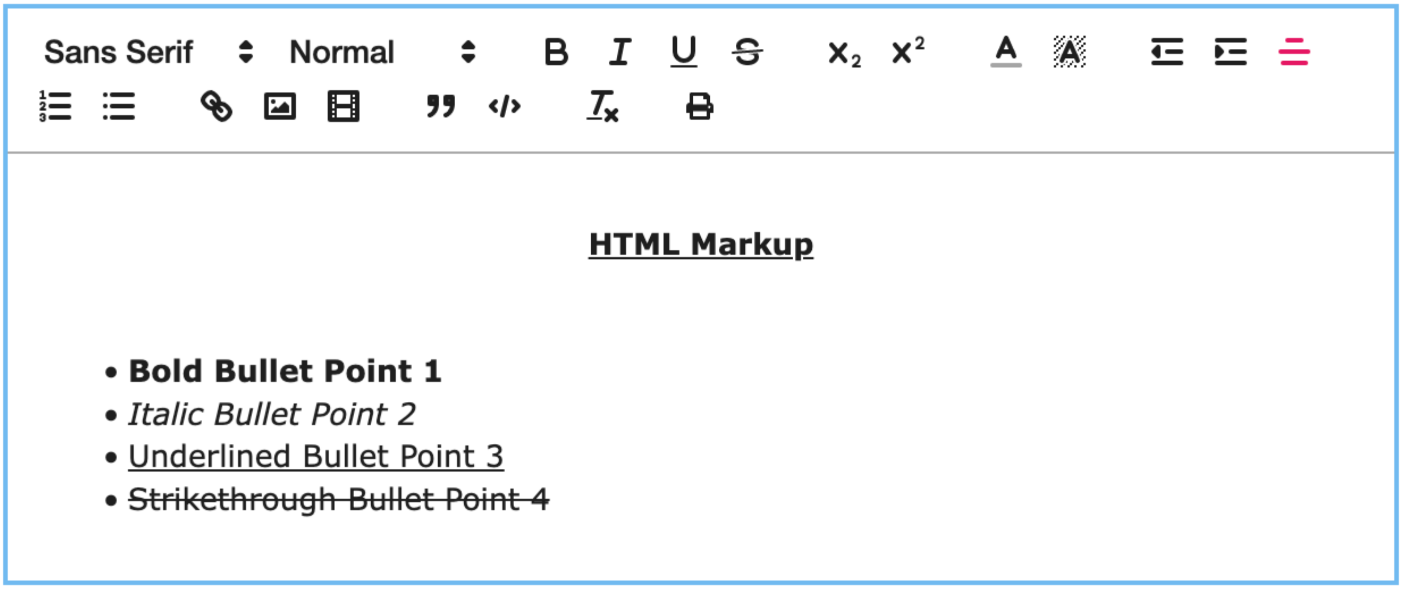
```
# $event method for Radiogroup
On evClick
  Calculate whereClause as con(
    'WHERE product_group =
    ',kSq,iGroupList.[pNewVal].product_group,kSq)
  Do $cinst.$objs.productList.$build(whereClause)
```

Rich Text Editor

Group	Icon	Name	Description
Entry Fields	B / U ≡ • Bold • Italic	Rich Text Editor	Rich text editor allowing end users and format text

The **Rich Text Editor** can be used instead of a regular Edit or Multi-line edit field, which adds the ability for end users to edit the text using a text editor UI and to apply “rich” formatting such as bold, italic, and simple bullets.

There is an example app called **JS Rich Text Editor** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.



HTML Markup

- **Bold Bullet Point 1**
- *Italic Bullet Point 2*
- Underlined Bullet Point 3
- ~~Strikethrough Bullet Point 4~~

Figure 201:

The Rich text editor control uses the *Quill* open source text editor which relies on modern browser technologies and, as such, some older mobile operating systems (iOS < 6.0 & Android < 4.0) may have compatibility issues. The control is based on Quill 1.0, which allows Code Blocks with syntax highlighting, Undo/redo shortcut keys, Sub/Superscript, In/Outdent, Block Quotes, Image uploads, content tips, and so on.

The text data for the control is stored in the instance variable assigned to $dataname: see below for properties to format the data content. You can allow text editing in the control by setting $showcontrols to kTrue where upon the text content in the field will become editable: on mobile devices this places the cursor in the field and opens the soft keypad ready for typing. The text editing controls in the field will appear at the top of the control and will allow the end user to format the text, including bold, italic, underline, and so on. The text data in the control is HTML markup and can include formatted text such as ordered (numbered) and unordered lists (bullets). End users can also insert images using the Paste option.

Dynamically Loaded Resources

The control dynamically loads the necessary JavaScript & CSS files when it is used. There are some files in the Studio tree to support the text editor:

```
html/scripts/
    quill.js
    highlight.pack.js
html/css/
    quill.snow.css
    highlight-theme.css
```

When deploying to a web server, you must make sure to also copy these files over.

Properties

Together with the standard component properties, the Rich Text Editor Control has the following properties:

Property	Description
$dataname	The name of an instance variable to store the HTML formatted text, or a column in an instance row variable

Property	Description
$dataformat	Controls the format of the document data stored in $dataname for the control, a constant: kJSRichTextDataFormatJSON, kJSRichTextDataFormatHTML, or kJSRichTextDataFormatPlain
$showcontrols	Set this to kTrue to switch the control to edit mode and to display text editing toolbar controls
$plaintextname	Specifies the name of a variable that automatically receives the plain text equivalent of the data stored in the variable named in $dataname (just the plain text without any HTML formatting); this property is optional
$contenttip	Allows you to specify some text to be displayed in the editor when it has no content
$removedtoolbaritems	A bitmask of kJSRichText… values, allowing you to specify toolbar items to hide in your Rich Text Editor instance

Setting Text Properties

You can specify the default value of the font, size and text color shown in the editor's controls by assigning values to the component's text properties, as follows:

- **$font**
  maps directly to the editor's font droplist and will set the default value accordingly.
- **$textcolor**
  will attempt to set the default text color to one of the colors in the toolbar's color palette. If there is not an exact match, it will add the color as another tile in the palette.
- **$fontsize**
  will set the default font size to the closest match. Set to **13** for the 'Normal' font size as default.

Data Format

The $dataformat property controls the format of the document data stored in $dataname for the control. It can be one of the following constants:

- **kJSRichTextDataFormatJSON**
  The document data will be stored in the $dataname as JSON, as a Quill 'Delta' object. This is the best option for restoring the data later, as it preserves all formatting.
  When setting the data, you can assign JSON (Delta), HTML or plain text: this should be detected and converted as necessary.
- **kJSRichTextDataFormatHTML**
  The document data will be stored in the $dataname as HTML. This may lose some minor formatting. This format is suitable to use if you are going to use the data elsewhere in your code, but not for storing the document data and restoring into the Rich Text Editor.
  When setting the data, you can assign HTML only.
- **kJSRichTextDataFormatPlain**
  The document data will be stored in the $dataname as Plain text. This will lose most formatting.
  When setting the data, you can assign Plain Text only.

The $dataformat property can be changed in your code to populate the $dataname with data of the specified format. Note that if you do this in a server-executed method, the $dataname won't be updated until the client next contacts the server.

Inserting Data

The $appenddata(cData, bNewLine) and $prependdata(cData, bNewLine) methods allow you to insert data in cData at the end (append) or beginning (prepend) of the content in $dataname. If you pass bNewLine as kTrue, the data will be added on a separate line. These methods can only be executed on the client.

The data format of the passed data depends on the value of the $dataformat property. If $dataformat is JSON, the data could be sent as plain text, HTML or JSON (a Quill Delta object).

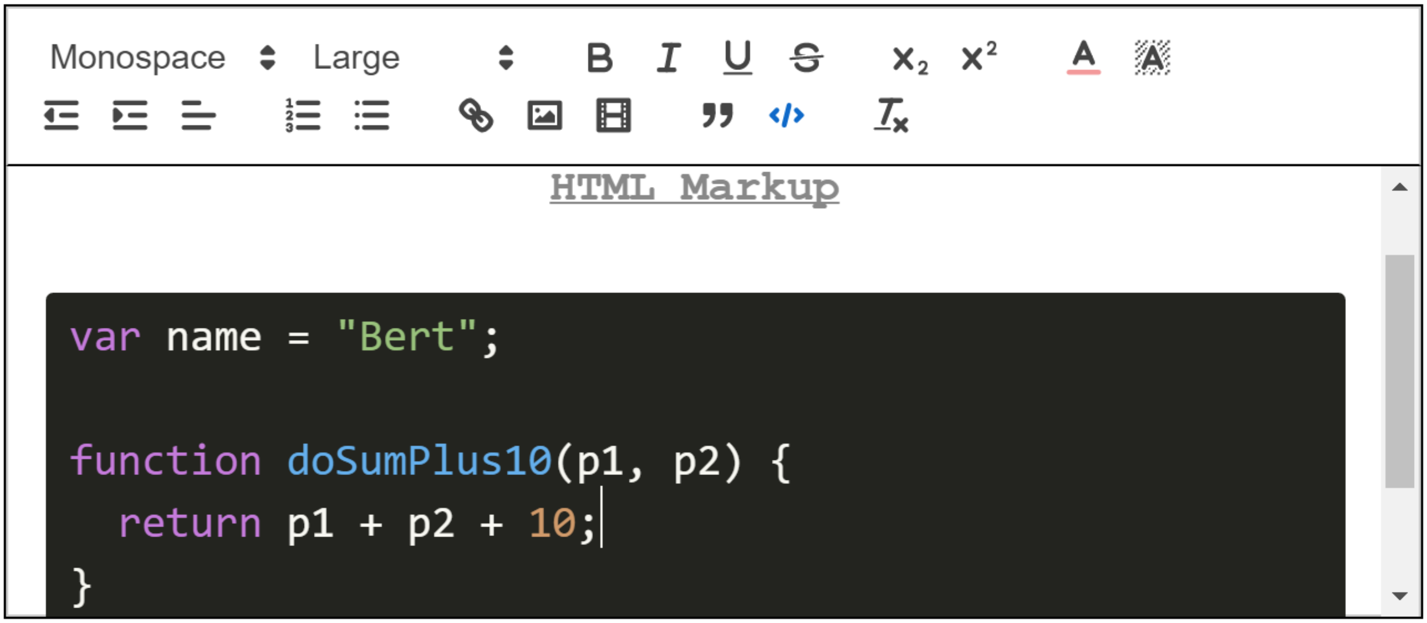
The $insertatcursor(cData) method allows you to insert the supplied text data in cData at the position of the caret within the Rich Text Editor the last time it had focus. The data can be plain text, HTML, or JSON (depending on the setting of $dataformat), and like the methods $appenddata() and $prependdata(), it must be executed on the client.

Getting Html

The **$gethtmlwithstyles()** client method returns the HTML from the Rich Text Editor and applies some inline styles to the elements, which when viewed externally, such as in a web browser, should closely represent the styles written in the Rich Text Editor. Note that there may be circumstances where the style is not exactly matched due to the limitations of inline vs stylesheet styling.

Code Blocks

The Rich Text Editor allows you to insert **Code Blocks**. These allow you to insert syntax-highlighted code. The syntax highlighting is achieved using highlight.js and by default includes highlight support for several popular languages.



HTML Markup

```
var name = "Bert";

function doSumPlus10(p1, p2) {
  return p1 + p2 + 10;
}
```

Figure 202:

If the language(s) you require is/are not supported out of the box, you can create a ‘Custom Package’ on the highlight.js download page, and replace the **highlight.pack.js** in your Omnis tree/web server with the one you download.

Similarly, if you want to change the code block’s appearance, you can take any of the theme css files from your highlight.js download, rename it **highlight-theme.css** and replace the supplied file with your own.

Drag and Drop

End users can drag data from a Rich Text Editor and drop it elsewhere in the form, or users can drag data and drop data into the Rich Text Editor; in the latter case the evDrop event is generated in the control. If some text is selected and dragged out of the Rich Text Editor, then only the selected text is dropped at the cursor position.

End users can also drag *external content* and drop it onto the Rich Text Editor, e.g. text or an image from another browser pane or a different application, but in this special case the evDrop event is not generated in the control.

Localizing the Rich Text Editor

There are various strings in the Rich Text Edit control that can be localized in the string table for the remote form containing the control. You must use the following string table ids to replace the default text for the controls in the text editor.

Tooltips for buttons

rt_backgroundcolor	rt_insertorderedlist	rt_print
rt_blockquote	rt_insertunorderedlist	rt_removeformat
rt_bold	rt_italic	rt_strikethrough
rt_clearformat	rt_justifycenter	rt_subscript
rt_codeblock	rt_justifyfull	rt_superscript
rt_decrease_indent	rt_justifyleft	rt_textalign
rt_image	rt_justifyright	rt_textcolor
rt_increase_indent	rt_link	rt_underline
rt_indent	rt_outdent	rt_video

Text displayed on controls

rt_fontsize	rt_fontfamily	rt_sansserif
rt_serif	rt_monospace	

Printing the Text Contents

End users can print the contents of the editor control using a print button on the editor's toolbar, which when clicked opens a window for printing. The $printcontents lets you print the contents.

- **$printcontents**(cTitle)
  Opens a new window to print the editor's current contents. cTitle is the title of the document to print.

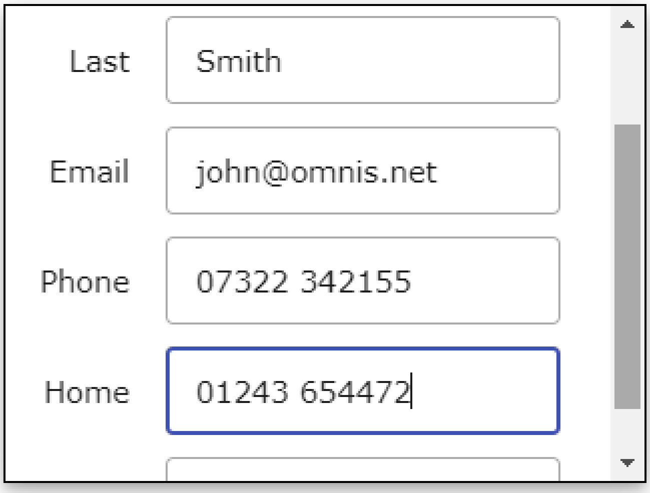
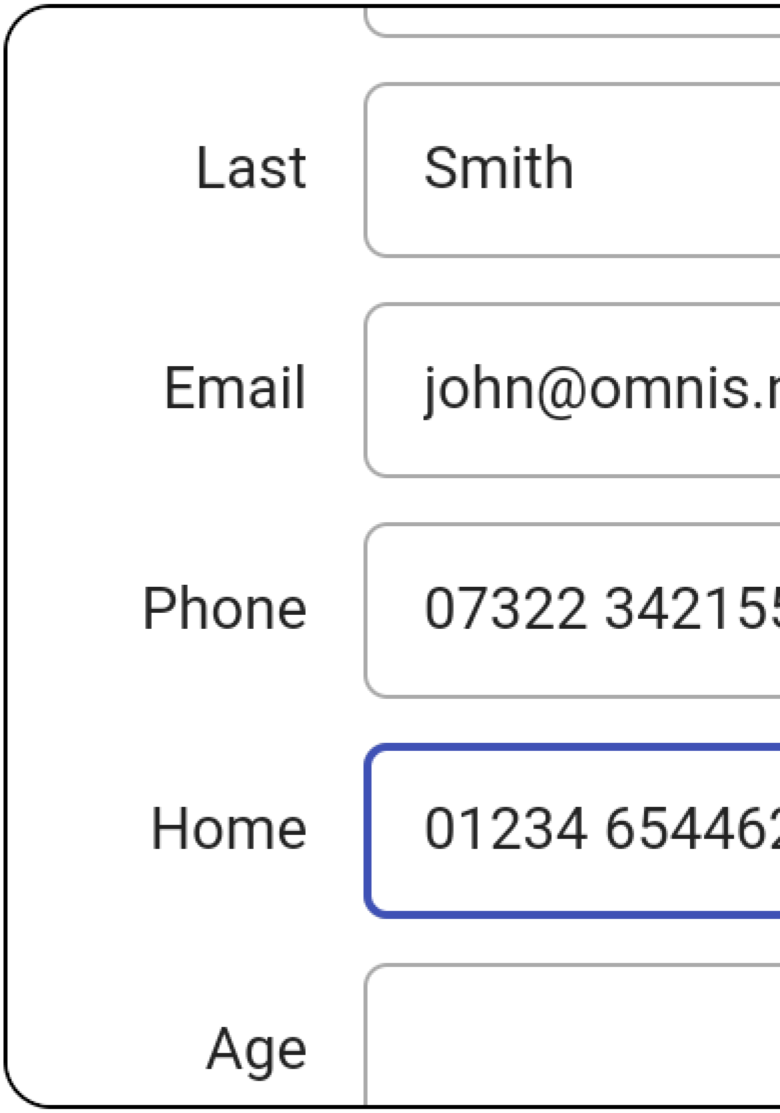
You enable the new print button by setting $removedtoolbaritems to kJSRichTextPrint. In addition, the Omnis string table item with ID: rt_print lets you edit the tooltip of the button.

Scroll Box

Group	Icon	Name	Description
Containers	◫	Scroll Box	Allows you to group other controls option to display a scroll bar if the does not fit

The **Scroll Box** component allows you to group together other controls on your remote form with the option to display a scroll bar if the content does not fit the visible area.

To enable the scrolling behavior, scroll boxes have the $autoscroll property. If true, and the client is displayed in a desktop browser, the client displays scroll bars permanently when the content does not fit the box area (see below left). On mobile devices, the scroll bar will be shown automatically when the content needs to scroll or as the control is dragged by the end user (see below right).

$autoscroll = kTrue on desktop	$autoscroll = kTrue on mobile
 Last Smith Email john@omnis.net Phone 07322 342155 Home 01243 654472	 Last Smith Email john@omnis.n Phone 07322 34215 Home 01234 65446 Age

Scroll boxes are container fields so you can access the fields inside the box in your code using the container notation. A Scroll box can contain methods including a $event() method to detect events, but not evClick.

A Scroll box can act as a side panel by enabling the $sidepanel property and setting $sidepanelmode (see the section about Side Panels), or it can contain other controls configured as side panels.

Scroll boxes have the $borderradius property, plus you can set $effect to add a border style, such as kJSborderPlain.

Subform Sets

You can use a Scroll box as the parent of a subform set, by specifying the scroll box name as the parent parameter when creating the subform set. In addition, you can add a new object to a scroll box using $cinst.$objs.$add with the scroll box name as the parent of the new control.

Group Box

You can convert a Scroll box into a Group box using the $makegroupbox() method, which must be executed on the client, and can be called from $init for the form.

- Scrollbox.**$makegroupbox**(cLabel[,cFont,cFontSize,cTextColor])
  turns a Scroll box into a Group box with the specified cLabel.

You can specify the font, size, and color in the cFont, cFontSize and cTextColor parameters (you can use CSS syntax). (Note the same method can currently be used to turn a Paged pane into a Group box.)

Alternatively, you can use the new properties $label, $labelfontsize, $labeltextcolor & $font to turn a scroll box into a group box at runtime, rather than using the $makegroupbox() client-executed method.

Setting the $label property for a Scroll box adds the label inside the border at the top of the control, effectively moving the top edge of the border down so that the label appears within the bounds of the control.

As with other controls with the $label property, you can double-click on the label text in design mode to edit the text.

Segmented Bar

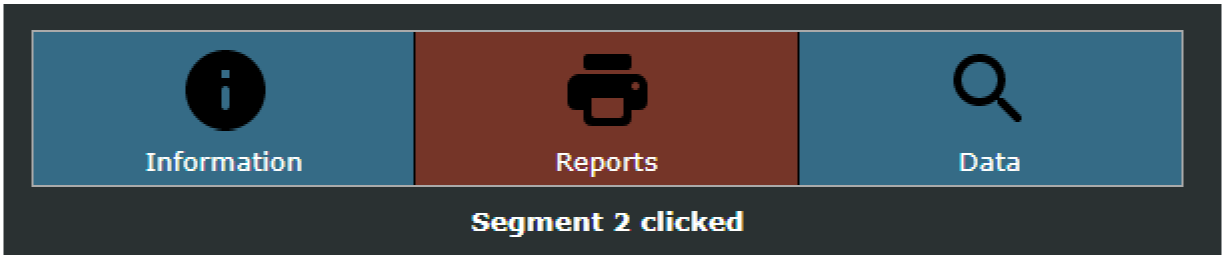
Group	Icon	Name	Description
Navigation		Segmented Bar	Navigation control with different b “segments”

The **Segmented Bar** control displays a number of buttons or “segments” that you can use for navigation (like a tab bar or toolbar) within your web or mobile app, or you can use it with only two segments to create a switch. You can assign an icon and text to each segment, and you can detect which segment has been clicked in the $event method of the control.

The Segmented control provides a series of ‘segments’ or buttons arranged horizontally, each of which can contain an icon and/or text. You can optionally show the selected segment in a highlighted state, by setting $showselectedsegment to kTrue, which is useful if you are using the segmented control as a navigation control. You can further show the current segment by the setting the color of the current segment indicator in the $focusedsegmentindicatorcolor property.

You can use the segmented control as a toolbar, docking it to the top or bottom of its container by setting its $edgefloat property to one of the kEFposn… values, such as kEFposnMenuBar or kEFposnTopToolBar.

There is an example app called **JS Segmented Control** in the **Samples** section in the **Hub** in the Studio Browser, showing how you can use the Segmented control as a toolbar and switch (the second image below); the same app is available in the JavaScript Component Gallery. The following images show the ‘professional’ JS Theme in use.



Segment 2 clicked

Figure 203:

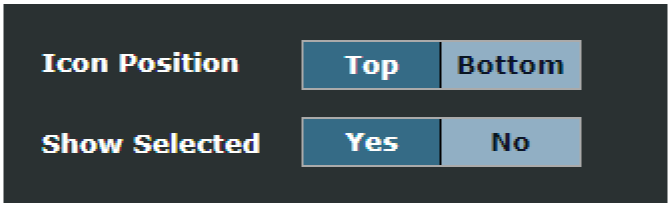


Figure 204:

Properties

The Segmented control has the following properties, together with the standard properties for a JavaScript control.

Property	Description
$currentsegment	The number (1 - $segmentcount) of the current segment (this specifies the segment affected by segment specific properties). This can also be changed in a design view by clicking on a segment of the design component. The current segment will be shown with a red outline while the component is selected
$hidedisabledsegments	Hides any disabled segments
$movesegment	Lets you move a segment (in design mode only)
$segmentcount	The number of segments (must be at least one)
$segmentenabled	If true, the segment is enabled and generates a click event when the user presses it.
$segmenticonid	The icon displayed on the current segment. Set to 0 for no icon
$segmenttext	The text displayed on the current segment
$displaystyle	A kJSSegmentStyle... controls whether the text is above or below the icon
$showselectedsegment	If true, the currently selected segment will be shown in a highlighted state. See $selectedcolor & $selectedtextcolor. If false, the highlighted appearance will still be shown while segments are being clicked, to give the user feedback of the click
$focusedsegmentindicatorcolor	Specifies the color of the current segment indicator
$segmentbordercolor	The colour that applies to borders / dividers of segments
$segmentborderradius	Single value border radius that applies to segments. If $segmentspacing is zero, this applies to only the outer edges of the outer segments. Otherwise, it applies to all segments
$segmenteffect	Determines whether borders / dividers are applied to segments, either kBorderNone or kBorderPlain
$segmentenabled	Set to kFalse to disable a segment
$segmentspacing	The space between the segments in pixels. The behavior can be affected by $segmentwidth (see below). If zero, dividers are drawn between segments. Otherwise, borders are drawn around the segments
$segmentwidth	The width applied to all the segments in pixels. By default, this is zero, in which case the width of the segments is determined by the total width of the control and $segmentspacing. If this value is small enough, the segments will be centred in the control
$selectedsegment	The number (0 - $segmentcount) of the currently selected segment. If 0 no segment will be selected
$selectedcolor	The background color of the currently selected segment, or of the segment currently being clicked
$selectedtextcolor	The text color of the currently selected segment, or of the segment currently being clicked

Property	Description
$bordercolor	Controls the color of the segment divider lines, as well as the control's border
$backcolor	Controls the background color of the segments

Segment size and spacing

If $segmentwidth and $segmentspacing are set so that the segments extend beyond the width of the control, the overflowing content will be scrollable. However, if $segmentwidth is zero (the default), the segments will always fit inside the container.

In the extreme case where $segmentspacing is very high, as long as $segmentwidth is zero, the spacing will be limited to prevent the segments becoming too small or the content overflowing.

Hiding Disabled Segments

You can set $segmentenabled for a segment to false to disable it. The property $hidedisabledsegments allows you to hide any segments that have been disabled.

Moving Segments in Design mode

The $movesegment property lets you move a segment: you need to set it to a number corresponding to the new position (the property works in the same way as the Data Grid's $movecolumn property).

Events

An evClick event is generated when one of the segments is clicked and the pClickedSegment event parameter returns the number of the segment clicked.

Slider Control

Group	Icon	Name	Description
Other		Slider	Slider component for setting value

The **Slider Control** provides a graphical slider component to set the value of a variable, such as a volume control, or to control the value of another component in your form. You can change the thumb icon if required. As an alternative, you can use the Native Slider to control variable numeric values.

There is an example app called **JS Slider** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

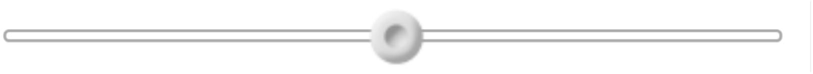


Figure 205:

Properties

The current value of the slider is reported in the property **$val** according to where the slider is positioned. You can specify the range for the slider in the **$min** and **$max** properties, while **$step** is the size of each step the slider takes between the min and max values.

When true, the **$reversescale** property swaps the $::max and $::min values on the scale of the slider. Therefore, when $::vertical is false (the default horizontal state), the min and max values on the slider are swapped left to right. When $reversescale and $::vertical are true, the min value is at the bottom, the max value is at the top.

The slider also has these properties:

Property	Description
$::vertical	If kFalse (the default) the slider is horizontal with $min and $max values shown left to right. If true, the slider is vertical
$sliderhorziconid	The id of the icon to use for a horizontal slider handle
$sliderverticonid	The id of the icon to use for a vertical slider handle
$horzmargin	The horizontal margin
$vertmargin	The vertical margin

Events

The Slider reports three events: evStartSlider (when the control is starting to track), evEndSlider (when the control has finished tracking), and evNewValue (when the value has changed). You can detect these events in the $event method for the component. These events all pass the current value of the Slider in the pSliderValue parameter. As the user drags the Slider thumb the evNewValue event is triggered and pSliderValue is sent to the $event method for the Slider.

To use the values of the slider in your remote form you can trap the slider events in the $event method of the slider control (which must execute on the client), and transfer the current values to instance variables in your form, as follows:

```
On evStartSlider
  Calculate iStartValue as pSliderValue
On evEndSlider
  Calculate iEndValue as pSliderValue
On evNewValue
  Calculate iNewValue as pSliderValue
```

Split Button

Group	Icon	Name	Description
Buttons	OK▾	Split Button	A button with a droplist of alternat options

The **Split Button** control combines a standard button with a dropdown menu, allowing you to provide multiple, alternate actions grouped together in a single button control. The Split Button is like the Send button in gmail as it provides two options in one control: a default *Send* option on the button and a *Schedule send* option via the menu.

The component is available for JavaScript remote forms as well as window classes, but there are some additional properties for the JavaScript control (the window version is an external component that must be loaded via the Component Store).

The menu for the control is specified in the $menuname property and must be a Remote Menu class for the JavaScript component, or a menu class for the Window class control.

The following example Split button has a Print option and a printer icon on the main button part, and it has options for printing to a Preview, PDF or File specified in a Remote menu class, specified in the $menuname property of the button control. In this case, a single click on the button would activate the Print to printer option, while clicking on the down arrow provides the other options (the image uses the 'professional' JS Theme).

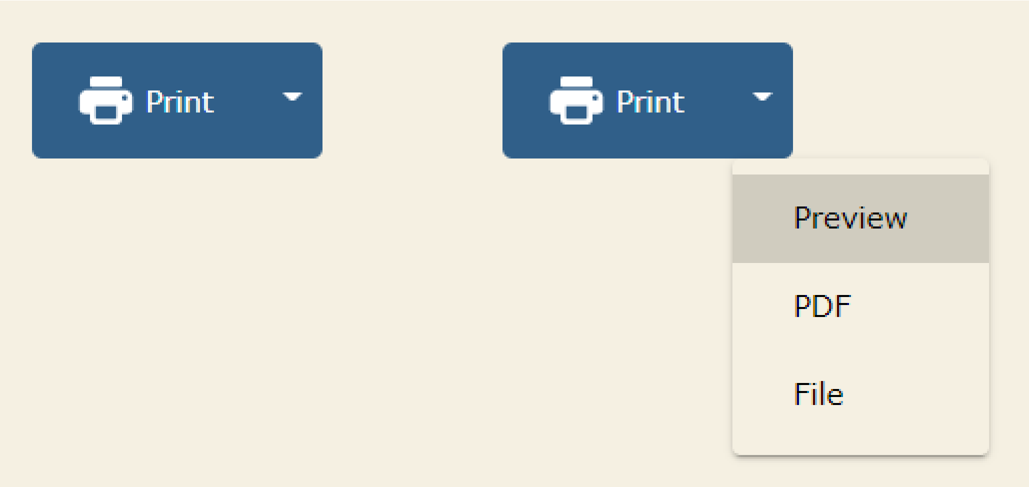


Figure 206:

Properties

The following properties are available for the Split Button.

Property	Description
$hotbackcolor	The background color of the control when hovered
$activebackcolor	The background color of the control while pressed; active color is generated automatically if $activebackcolor is kColorDefault
$buttonborderradius	The radius in pixels of the corners
$borderwidth	The width (0-7) of the edges drawn as the border of the control
$arrowside	The position of the dropdown button on the control
$textbeforeicon	If true, and the control has both text and an icon, the text is drawn before the icon
$vertical	If true, the text and icon are arranged vertically
$menuname	The name of the menu class, a Remote Menu class for the JS control, or a Menu class for the Window control
$menubackcolor	The background color of menu lines (JS only)
$menuhotbackcolor	The background color of menu lines when hovered (JS only)
$menutextcolor	The text color of menu lines (JS only)
$menuhottextcolor	The text color of menu lines when hovered (JS only)
$menudisabledtextcolor	The text color of disabled menu lines (JS only)

Events

An evClick event is triggered when the main button area is pressed. In addition, for the JavaScript client only, the evOpenContextMenu and evExecuteContextMenu events are generated when the menu is pressed, and in this case, the pControlMenu event parameter is kTrue (when a Context menu is opened pControlMenu will be kFalse), and the pCommandID parameter contains the command ID of the selected menu line, e.g. 1002 for line 2.

Example

There is an example app called **JS Split Button** in the **Hub** in the **Studio Browser,** and the same app is available in the JavaScript Component Gallery. The example library contains a JS remote form and remote menu class (plus a window class and menu to show the thick client split button). The following image shows the JS Split button example using the 'default' theme.

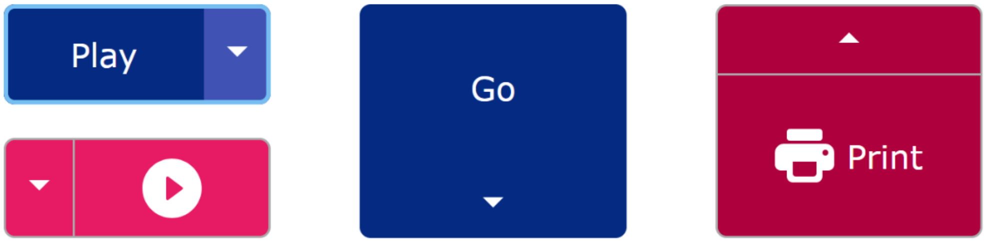


Figure 207:

The $event method behind the split button control traps the evClick and evCCC events corresponding to clicks on the button or the menu part of the control.

```
On evClick
  Do method buttonClicked
On evExecuteContextMenu
  Do method menuClicked (pCommandID)
# buttonClicked method
Do $cinst.$showmessage('Button Clicked!','Split Button')
# menuClicked method
Do $cinst.$showmessage(con('Remote Menu Clicked!//pCommandID=',pMenuLineCommandID),'Split Button')
```

The pCommandID parameter is passed to the menuClicked method which contains the command ID for the selected menu line ($commandid is a menu line property which stores IDs 1001, 1002, etc for successive menu lines).

Subform Control

Group	Icon	Name	Description
Subforms	▣	Subform	Allows you to display another remo class as a subform in the main form can create a subform set)

The **Subform** control allows you to place another remote form class inside the main remote form. The concept of the subform is similar to embedding an iframe into an HTML page where you can embed another page or form or inside the main page. You could, for example, create a single "main" form and a number of other remote forms loaded at runtime into a subform control, to create a powerful and interactive web application with many subforms. For example, the JavaScript Component Gallery app on the Omnis website is implemented using a main gallery form and each example component is loaded in a separate form using the subform control.

When you have placed the subform control on your remote form, you specify the initial remote form to appear in the subform in the $classname property. Alternatively, you can switch subforms at runtime by assigning a new remote form name to $classname to switch the current form displayed in the subform control, as follows:

```
# oSub is the name of the Subform control on the main Remote form
Calculate $cinst.$objs.oSub.$classname as NewFormName
```

Note: you can also create a group of subforms dynamically in your code using a **Subform Set,** which is described in the Subform Sets section.

There are a few example apps showing how you can use subforms or subform sets in the **Samples** section of the **Hub** in the Studio Browser (called **JS Subform, JS Subform Set, JS Subform Set Panels**), and the same apps are available in the JavaScript Component Gallery.

$construct and Subforms

Opening a remote form containing a subform field or any number of subform objects creates an instance of each form, which belong to the same task as the parent remote form instance. Omnis calls the $construct() method of *the parent form instance first,* then the $construct() methods of *all the subform classes in tabbing order* are called. The reverse happens on closing the parent form, with the subforms being destructed before the parent form instance. Note that the $construct() of a subform on a hidden page of a Paged Pane will not be called until the page is shown.

You can send parameters to the subform's $construct() method by including a comma-separated list of parameter values in the $parameters property when you create the subform field. The $parameters property can only be assigned in the remote form editor, and it has no effect at runtime. If you attempt to set it at runtime in your code, the error "$parameters cannot be assigned at runtime" will be displayed on the client.

$init and Subforms

When changing subform instances you can send parameters to the new target subform instance. To do this you can assign a comma-separated list of values to the $userinfo property of a subform and this can be parsed and sent as parameters to the client-executed $init method in the new subform instance. Each token in the comma-separated list will be a separate parameter, and can be a quoted string (including spaces and commas) or a numeric value. For example, when changing subform:

```
Calculate $cinst.$objs.subform1.$userinfo as "'Davy Jones',123,0"
Calculate $cinst.$objs.subform1.$classname as "jsSubform1"
```

In the $init method in the new subform jsSubform1, three parameters will be populated: p1: "Davy Jones", p2: 123, p3: 0.

The $loadfinished client-executed method allows you to check when all subforms of a form have been loaded. The method is called after all the subforms that belong to the parent remote form instance have finished loading and their $init methods have been called.

Multiple Subforms and Caching

The $multipleclasses property tells Omnis to keep a set of remote form instances open for use in the subform object, rather than constructing a new instance each time the subform class is changed. When you assign a new remote form name to $classname at runtime, the new remote form is downloaded to the client and displayed in the client's browser. If the $multipleclasses property is enabled, the previous remote form is cached and hidden on the client, otherwise the remote form instance is destroyed. If any previous remote forms have been cached in this way using $multipleclasses, you can switch back to them instantaneously, otherwise they have to be reloaded each time you assign to $classname of the subform object.

Layout Breakpoints

It is not required that all subforms within the inheritance hierarchy of a set of Remote forms have the same layout breakpoints. In other words, a subform can have different layout breakpoints to its superclass.

Referencing Subform Instances

A subform control has the $subinst property (object) which is the instance contained within the subform control. You can use this property to get a reference to the instance in a subform object and therefore change properties within the instance. If the subform property $multipleclasses is set to kTrue, you must use $subinst(cClassName) to get a reference for the appropriate instance. For example, where a subform control has a single subform class the following code will return a reference to the form instance in the subform:

```
Set reference item to $cinst.$objs.subfrm.$subinst
Do item.$setcolor(kRed)
```

or when you may have assigned multiple classes to the subform control ($multipleclasses is set to kTrue):

```
Set reference item to $cinst.$objs.subfrm.$subinst("classname")
Do item.$setcolor(kRed)
```

Note that the item returned will be null if the instance does not exist.

Subform Container Notation

You can use $cinst.$container to obtain the instance containing a subform instance, where in the current context $cinst is a subform. The $container property returns the object containing the referenced object. This notation will work in server methods (for all clients) , and client-side methods for the JavaScript Client only. For examples:

```
Set reference item to $cinst.$container
Do item.$getList() Returns iList
# and
Calculate $cinst.$container().iSelected as iList.C1
```

Subform Promise

You can return a promise from client-side assignments to subform $classname. The promise will be resolved when the form is loaded, after its $init has run. When assigning the $classname of a subform at runtime you can return a promise, e.g.

```
Do $cinst.$objs.sf1.$classname.$assign("sub2") Returns lPromise
JavaScript:lPromise.then(() => {
Do $cinst.$objs.sf1.$subinst().$whatever()
JavaScript:});
```

Rather than rejecting the promises when an error occurs, an error message is passed as the first parameter to the resolve function. If this is populated, you can treat it as an error.

Events

The **evSubformLoaded** event is triggered when the form instance in a subform control changes, after the $init method in the form instance has been called. The event is also triggered if $multipleclasses is true and an existing form is being switched back to.

The evSubformLoaded event receives two parameters:

- **pFormName**
  The name of the Remote Form class which has just loaded.
- **pInitialLoad**
  True if the form instance has just been constructed. It may be false if $multipleclasses is true, and an existing Remote Form is coming back to the front.

Example

The **Holidays** sample app (in the **Applets** section of the **Hub** in the Studio Browser) uses a subform to display either the User or Admin form. When the main jsHolidays remote form is loaded, its $construct() method calls a class method to setup the initial subform to be shown which, in this case, is the User form. In addition, there are buttons on the main Holidays form to allow the end user to switch forms; for example, the code behind the User button is:

```
# $event method for User button on jsHolidays
On evClick
  Do method setSubForm (kUserForm)
  Calculate iAdminBtnState as kTrue
  Calculate iUserBtnState as kFalse
  Do $cinst.$objs.adminBtn.$enabled.$assign(kTrue)
  Do $cobj.$enabled.$assign(kFalse)
```

The setSubForm method has the following code to switch forms:

```
If pOption=kAdminForm   ## test if Admin or User
  Do $cinst.$objs.subForm.$classname.$assign("jsAdminForm")
Else
  Do $cinst.$objs.subForm.$classname.$assign("jsUserForm")
End If
```

Switch Control

Group	Icon	Name	Description
Buttons		Switch	Allows on/off selection; you can spe icon for on/off state

The **Switch Control** is like a check box insofar as it represents an On / Off value (1 or zero), but it can display alternative images for the On or Off states. When the end user clicks the switch the value of the control's variable alternates between 1 and zero, so the control is useful for representing preferences which can be Enabled or Disabled, or items that can be turned On and Off, like a switch.

The variable you specify in the $dataname property should be a Number or Boolean variable. The $switchon and $switchoff properties let you specify an icon ID for the images to be used when the switch is either On or Off. The properties $justifyhoriz and $justifyvert allow you to justify the contents in the control horizontally or vertically.

The $switchcolor property specifies the color for the Switch control when it is switched on (set to value 1), assuming no on/off icons have been set.

Alternatively, you can use the Native Switch component to display on/off values.

Example

There is an example app called **JS Switch** in the **Samples** section of the **Hub** in the Studio Browser, showing the Switch control with a range of different ON/ OFF images; the same app is available in the JavaScript Component Gallery.

Figure 208:

Tab Bar Control

Group	Icon	Name	Description
Navigation	Tab1 Tab2	Tab Bar	Multiple tabs to control the selecti page pane

The **Tab Bar** control (or Tabbar) allows the end user to select a tab which can correspond to a specific option in your application; the tab control can also be linked to a Paged Pane by setting $linkedobject. The $tabcount property specifies the number of tabs; it can be set to zero in design mode, and the number of tabs assigned at runtime using the notation. The $currenttab property specifies which tab is highlighted and its value will change as the end user selects a tab: assigning a new value to this property at runtime will change the highlighted tab; the following screen shows the Tab control using the 'health' JS Theme.

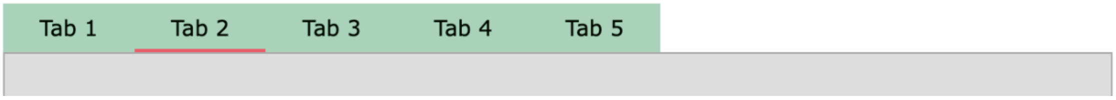


Figure 209:

In design mode, you can specify the properties for a particular tab by making it the "selected tab" in the $selectedtab property, under the "Tab" tab in the Property Manager. The text for a tab is specified in the $tabtext property: you can add a line break by inserting //. The size of the tab (i.e. the width for horizontal tabs) is determined by the width of the text on the tab. However, you can set $fixedtabsize to kTrue to fix the size of the tabs, and set $maxfixedtabsize to set the tab width or height. You can hide or show a tab using the $tabvisible property, and you can disable or enable a tab using $tabenabled; for example, you can set these properties for individual tabs to kFalse in design mode and at runtime set these properties to kTrue to show and enable the tabs.

There are many properties under the Appearance tab in the Property Manager to control the general appearance of the Tab Control and the tabs themselves. The tabs have angular corners by default, but you can round the corners by setting $tabborderradius. You can create a vertical aligned set of tabs by setting the $side property: you can orient the tabs on the left or right. The $tabsjst property determines the position of the tabs within the control. The $currenttabindicatorcolor property specifies the color of the current tab indicator.

The $tabbackstyle property controls the style or color of the background of the tabs and is a kJSTabsBackStyle… constant: kJSTabsBackStyleDefault shows a default appearance for the current platform, kJSTabsBackStyleColor shows a flat background color specified in $tabbackcolor, and kJSTabsBackStyleImage displays an image specified in $tabbackiconid.

There is an example app called **JS Tab bar** in the **Samples** section in the **Hub** in the Studio Browser showing how you can link a tab control to a paged pane, as well as using a tab control to display a menu; the same app is available in the JavaScript Component Gallery.

Tab Icons

You can specify an icon for a tab under the Tab tab in the Property Manager (shown when the Advanced option is enabled); you can set the $selectedtab property to select a tab in design mode. You can set the position of the icon relative to the text by setting $tablayout. For example, you can set $tablayout to kJSTabsLayoutIconLeft to add an icon to the left of the tab text, and then set $tabiconsize to set the width of the space allowed for the icons.

In addition to the icon you can assign to a tab, you can add 'Icon Badges' to a tab icon to provide additional information, such as a number count, a notification, or an alert: see Icon Badges.

Reordering Tabs

In design mode, you can move or re-order the tabs in the control by entering a number into the $movetab property; in effect, the number of the selected tab will become the number you entered, and the other tabs are shuffled along. This is useful if you have setup multiple tabs and need to move a tab easily without having to redefine each tab again.

Tab Pane

There is a **Tab Pane** control in the **Containers** group in the Component Store that is a compound object containing a Tab Control and a Paged pane linked together, as described below.

Linking Tabs to Panes

The Tab Control can be linked to a Paged Pane by setting $linkedobject to the name of a Paged Pane control, so when different tabs are clicked, the pane in the linked Paged Pane is changed automatically. Assigning a new value to the $currenttab tab property of the Tab control at runtime will also change the current pane in the linked Paged Pane control.

The base edge of the tab control does not normally have a border, and when the tabs are linked to a paged pane the tab control updates the paged pane border so that there is the appearance of a gap below the current tab. If you want to add a border under the tabs you need to set $baseedgewidth to 1 or more.

Tab Menus

The Tab Control can also be linked to a Remote Menu class; clicking on a tab will trigger the corresponding line in the menu. To implement this, the $trackmenus property must be kTrue, and the $tabmenu property set to the name of a remote menu class for the selected tab (the tab with number $selectedtab). If assigned at runtime, the menu instance must already be present on the client (via a $tabmenu or $contextmenu property in the class data when the form was loaded).

You can control the color of the menu lines and text using the $tabmenu… properties under the Appearance and Text tabs in the Property Manager, or in your code for the control.

Events

When the end user clicks on a tab the value of $currenttab will change and an evTabSelected event is triggered, with the new tab number reported in pTabNumber. This event needs to be enabled in the $events property for the control to be reported.

The $canclickselectedtab property can be enabled so a click on the selected tab generates evTabSelected (provided that evTabSelected is specified in $events). This allows you to detect a click on the currently selected tab (which was not possible in previous versions).

Example

The **Contacts** sample app users a Tab control in the main jsContacts remote form to allow the end user to switch from viewing a list of contacts to a form showing details of individual contacts. In this case, the Tab control is linked to a page pane which displays the contact list or contact details view. The $linkedobject property of the contactTabStrip control is set to 'pagePane' (the name of the page pane) and the $tabtext for each tab is defined as 'Contacts' and 'Details' respectively. In addition, the evTabSelected event is enabled in the $events property of the Tab control. The code for the Tab control $event method is:

```
# $event method for Tab control
On evTabSelected
  If pTabNumber=2
    Do method loadRecord (iContactList.$line)
    Do $cinst.$objs.saveBtn.$enabled.$assign(kTrue)
  Else If pTabNumber=1
    Calculate iNewContact as kFalse
    Do $cinst.$objs.saveBtn.$enabled.$assign(kFalse)
  End If
```

If the Details tab is clicked, the second tab, the second pane in the page pane is displayed and the details for the currently selected contact are loaded using the loadRecord class method.

Tile Grid

Group	Icon	Name	Description
Lists	▦	Tile Grid	Displays a scrollable grid of tiles wh be configured to show images, tex buttons

The **Tile Grid** component displays a scrollable grid of tiles which can be configured to show images, text and buttons. The layout of the grid and the visual attributes for the tiles are specified in a list variable which is assigned to **$dataname** of the control; each line in the list provides the definition for a single tile in the grid. At runtime, the tiles are loaded and unloaded dynamically as the grid is scrolled, to improve the UX and performance.

There is an example application called **JS Tile Grid** in the **Samples** section of the **Hub** in the Studio Browser which displays a number of tiles using images from the webshop example app, as follows:

Properties

The Tile grid has the following properties.

Property	Description
$dataname	List instance variable defining the tiles, see below

Property	Description
$centertiles	If true, and $tilefixedwidth is such that tiles do not use the full width, tiles will be centered.
$tilefixedwidth	The fixed width of tiles in pixels (default is 0). Takes priority over $tileminwidth and $columncount.
$columncount	The number of grid columns (default is 2); set to 0 for column count to be set automatically. Only applied when $tileminwidth and $tilefixedwidth are zero
$tileminwidth	The minimum width of tiles in pixels (default is 0); applied when $tilefixedwidth is zero
$tileheight	The height of tiles in pixels (default is 140)
$tilegap	The gap between tiles in pixels (default is 5)
$tileborderradius	The border radius used for tiles (default is 4)
$titlebarposition	The position of the title bar on the tile, a constant: kJSTileGridTitleBarPositionBottom (the default) kJSTileGridTitleBarPositionTop kJSTileGridTitleBarPositionNone
$titlebarlayout	The layout of the title bar and background image, a constant: kJSTileGridTitleCoversImage: Title bar covers the image (the default) kJSTileGridTitleBesideImage: Title bar is beside the image kJSTileGridTitleBesideImageAndBackground: Title bar is beside the image and background
$imagescaling	The scaling type for tile images, a constant: kJSTileGridScalingCover: Size image to cover the available space, maintaining its aspect ratio (the default) kJSTileGridScalingContain: Size image to fit inside the available space, maintaining its aspect ratio kJSTileGridScalingFill: Stretch image to fill the available space kJSTileGridScalingNone: Do not resize image
$titlebarheight	The height of the title bar in pixels (default is 60)
$titlebarcolor	The color of the title bar
$buttoncolor	The color of tile action buttons
$tilecolor	The default tile background color; can be overridden for individual tiles in the data list using the BackgroundColor parameter
$tilehotcolor	The default hovered tile background color; can be overridden for individual tiles in the data list using the HotBackgroundColor parameter
asof 35876 $tileshadow	Adds a shadow to the tiles in the grid

Property	Description
asof 35901 $tilegrowonhover	enables tiles to grow when the user's pointer hovers a tile
asof 35901 $tileimagezoom	zooms the image of a hovered tile, specified as the percentage by which the image expands
asof 35901 $horzpadding	specifies the left and right padding inside the tiles in the grid
asof 35901 $vertpadding	specifies the top and bottom padding inside the tiles in the grid
$text1align	The text alignment for the primary text field in the tiles
$text1color	The color used for the primary text field in the tiles
$text1font	The font used for the primary text field in the tiles
$text1size	The point size used for the primary text field in the tiles
$text1style	The font style used for the primary text field in the tiles
$text2align	The text alignment for the second text field in the tiles
$text2color	The color used for the second text field in the tiles
$text2font	The font used for the second text field in the tiles
$text2size	The point size used for the second text field in the tiles
$text2style	The font style used for the second text field in the tiles

Configuring the grid layout

The tiles are arranged in the Tile Grid control from *left to right* across the grid, wrapping onto successive lines according to the total number of lines in the source list and thereby the number of tiles to be displayed. You can set **$columncount** to specify a fixed number of columns across the grid, and in this case, the width of the tiles is adjusted automatically to fit the width of the grid control. Alternatively, you can set $columncount to zero and use **$tileminwidth** to specify the minimum width of the tiles (columns), so that the number of columns is set automatically depending on the overall width of the control, i.e. the number of columns is adjusted automatically as the control is resized in a responsive form. If both properties are used, $tileminwidth takes priority.

Each tile in the grid can have an *action button,* which can be clicked by the end user, as well as *primary text* (e.g. a title) and *secondary text* (e.g. a description), which are placed inside a *title bar* positioned at the bottom or top of the tile. The tile background also responds to end user clicks.

When the whole tile grid has the focus after being tabbed to it, pressing the Enter key will put the focus on an element within the grid. From there, clickable elements can be tabbed through and activated with the Enter or Space keys. Pressing Escape will return the focus to the whole grid.

Setting the current line in the list will set the current tile and scroll the grid to that tile. The current tile is assigned a CSS class "ctrl-tg-current" to which you can apply custom styling in user.css, if required.

Setting the tile width

The width of the tiles in the grid can be specified by setting the **$tilefixedwidth** property; if specified, this takes priority over $tileminwidth and $columncount. In this case, the number of tiles (columns) that fit into the width of the control is calculated automatically from the value of $tilefixedwidth.

Alternatively, when $tilefixedwidth is set to zero, you can use $tileminwidth to set a minimum width for tiles, or when $tileminwidth *and* $tilefixedwidth are zero, you can use $columncount to specify the number of columns across the grid and in this case each tile will stretch to fit the available column width.

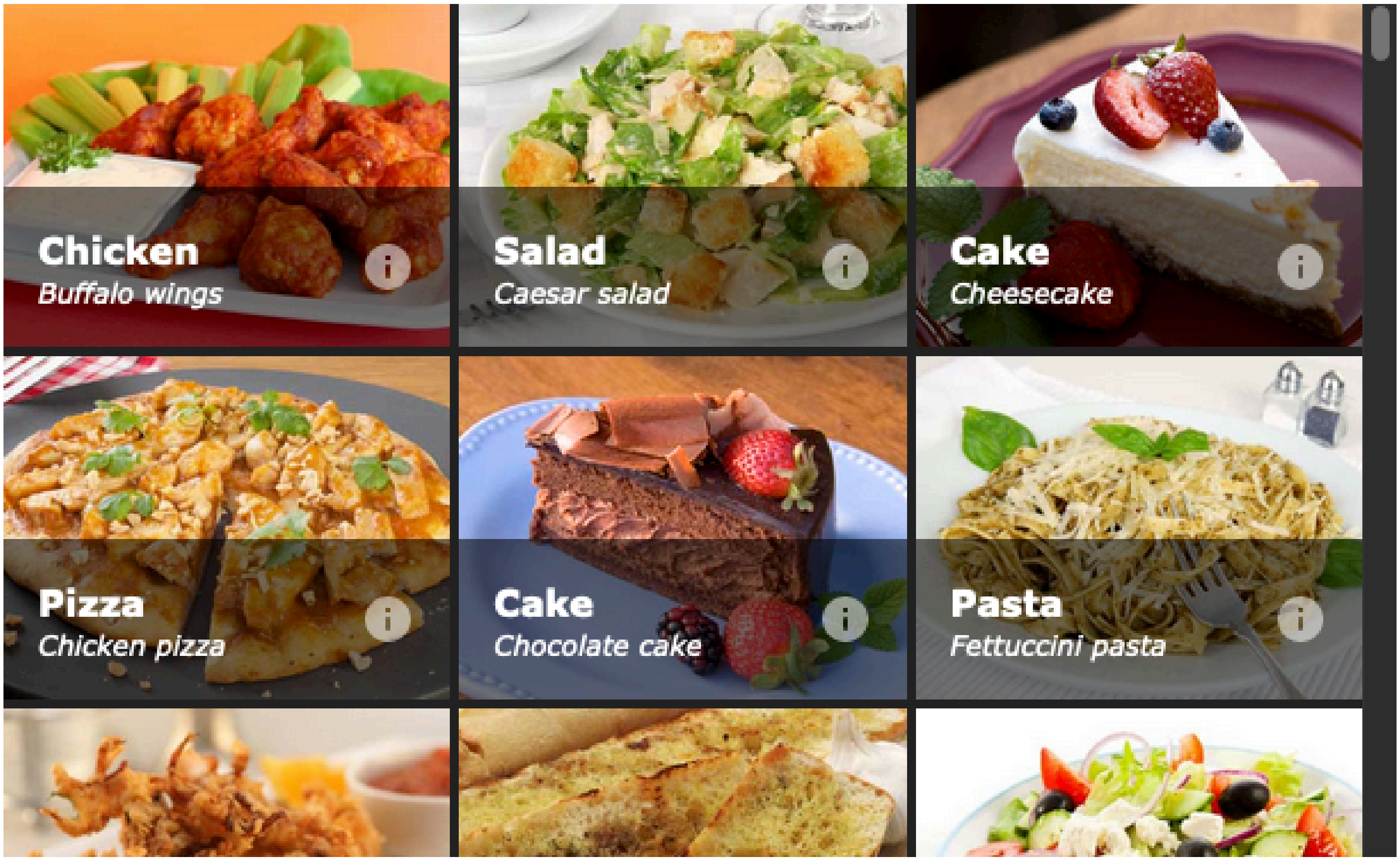

Chicken
Buffalo wings
Salad
Caesar salad
Cake
Cheesecake
Pizza
Chicken pizza
Cake
Chocolate cake
Pasta
Fettuccini pasta

Figure 210:

If $tilefixedwidth, $tileminwidth and $columncount are all zero, all tiles will fit into a single row.

The height of the tiles is set in $tileheight (the default is 140 pixels), while the gap between tiles is set in $tilegap (the default is 5 pixels).

Defining the data list

The list instance variable assigned to **$dataname** contains tile specific information, with each row in the list representing a single tile. The order of columns does not matter, and all columns are optional, but they must have the following names:

- **ImagePath:** The URL of the background image for the tile. If not specified or null, the tile's background color $tilecolor will be visible.
- **Text1:** The primary text or title to display on the title bar. Also used as the "aria-label" accessibility attribute, and the "alt" attribute of the image.
- **Text2:** The secondary text or description to display on the title bar.
- **ButtonPath:** The URL of the image for the action button. If not specified or null, no button will be added. iconurl() can be used to reference an icon in an icon set, e.g. iconurl("info") to show an info icon
- **ButtonDescription:** A description of the action button. If specified, this is the tooltip text, and "aria-label" accessibility attribute for the button.
- **BackgroundColor:** The background color of the tile.
- **HotBackgroundColor:** The background color of the tile when it is hovered.

For example, the following code from the example app (in the Hub) defines the list and adds a number of tiles:

```
Do iData.$define(ImagePath, Text1, Text2, ButtonPath, ButtonDescription, BackgroundColor)
Do iData.$add("images/webshop/BuffaloWings.jpg", "Chicken", "Buffalo wings", iconurl("info"), "Info", kJSTheme
Do iData.$add("images/webshop/Caesar_Salad.jpg", "Salad", "Caesar salad", iconurl("info"), "Info", kJSThemeCol
Do iData.$add("images/webshop/Cheesecake.jpg", "Cake", "Cheesecake", iconurl("info"), "Info", kJSThemeColorPri
# etc
```

In addition to using bitmap images (JPG or PNG), you can add an an SVG image from an icon set to the background of a tile. In this case, you can use the iconurl() function to reference the SVG image.

Events

The tile grid has two events: **evButtonClick** is sent when the action button for a tile is clicked, while **evTileClick** is sent when a tile is clicked anywhere except on the action button. The tile displays a ripple effect when it is clicked. For both events, the **pClickedTile** event parameter returns the index of the tile that was clicked, starting at 1 for the first tile in the grid.

Timer Control

Group	Icon	Name	Description
Other	⏱	Timer	Timer object triggers an event at a specified interval

The **Timer Control** is an invisible component that triggers an evTimer event after a specified time while $running is set to kTrue. You can specify a $timervalue, which is interpreted as an interval in seconds or milliseconds according to $useseconds, which should be set to kTrue for seconds or kFalse for milliseconds (the default).

There is an example app called **JS Timer** in the **Samples** section in the **Hub** in the Studio Browser, and the same app is available in the JavaScript Component Gallery.

Example

In the **Webshop** sample app (available under the Applets section in the Hub), it would be possible to use a Timer control to rebuild the Orders list periodically; in this case, you could use the Timer object to run a method at a given interval to rebuild the Orders list in the jsShopOwner remote form. In reality, the Orders list is rebuilt every time a new order is placed, but a Timer object could be used to regulate the rebuilding of the Orders list.

To implement a Timer in the Webshop app, you would need to enable the evTimer event in the $events property of the Timer control. The $useseconds property is set to kTrue and $timervalue is set to 2 (to give an interval of 2 seconds). The $construct() method of the Orders list (a data grid) includes a line of code to start the timer:

```
# $construct method for Orders list/data grid
Do iOrderList.$definefromsqlclass($tables.T_qOrders)
Do $cinst.$objs.timer.$running.$assign(kTrue)     ## possible using a timer object
Do $cfield.$build()
```

The $build() method behind the Orders list builds the list when the form is opened, but when used with the Timer object it can be used to rebuild the list as well. The $event method for the Timer object is run every 2 seconds, and has the following code:

```
On evTimer
  Do $cinst.$objs.dataGrid.$build()
```

Timer Worker Object

The timer component contains a Worker Object. This has the advantage over the other timer objects in that it can be used with remote tasks in the multi-threaded server. It has the following properties:

- **$timervalue** and **$useseconds**
  These work as for the current timer objects
- **$repeat**
  If true, then after calling $starttimer() the timer will fire until $stoptimer() is called or the object is deleted;otherwise the timer will fire at most once for each call to $starttimer().A change to $repeat is ignored until the timer is started again

The timer component supports the methods $starttimer() and $stoptimer(). Just like $repeat, changes to $useseconds or $timervalue do not take effect if the timer is already running.

When the timer fires (or the timer is cancelled), Omnis calls the $completed or $cancelled method in the object, just like other worker objects. This occurs in the context of the task that owns the object, and interrupts any code running for that task (after a complete method command has executed).

Toolbar Control

Group	Icon	Name	Description
Navigation		Toolbar	Toolbar with custom buttons (icon text), auto overflow and optional si

The **Toolbar** control allows you to add a series of buttons and an optional menu at the top or side of a remote form that the end user can click on or tap to perform an action. Each toolbar button can be assigned an icon and text, as well as a different action. When a button is clicked, the item number is reported to the event handling method allowing you to run the appropriate code.

The toolbar is displayed horizontally, by default, but can also be displayed vertically. You can use $edgefloat properties to 'stick' the toolbar to the top or side of the remote form. If items do not fit on the toolbar, they are added to an overflow menu automatically (shown by three vertical dots on the right or bottom of the toolbar), or items can be forced to appear on the overflow menu.

A toolbar can have a "side menu", displayed at the beginning (left or top) of the toolbar, by setting $sidemenu to true and adding a list variable name to $dataname containing the menu items. Items in the side menu can have a 'selected' state as well as a 'focused' state. Selecting a line in the side menu sets the current line in the list. The selected line will remain highlighted until another line is selected. When the side menu is opened, the selected line will get the focus.

To set the properties of each button or item, such as the text or icon, you can click on the button and set its properties under the Item tab in the Property Manager (shown when the Advanced option is enabled), or you can set the $currentitem property to select an item in design mode.

In addition to the icon you can assign to a toolbar button, you can add 'Icon Badges' to toolbar button icons to provide additional information, such as a number count, a notification, or an alert: see Icon Badges.

Example

There is an example app called **JS Toolbar** in the **Samples** section in the **Hub** in the Studio Browser showing a Toolbar with overflow and a side menu; the same app is available in the JavaScript Component Gallery.

The example Toolbar has four items or buttons defined, each with an icon and text, the main title 'Toolbar' on the left, and an overflow menu on the right.

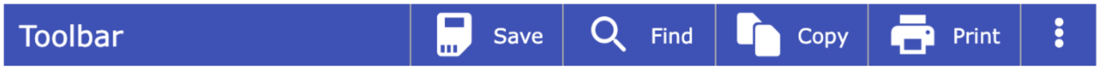


Figure 211:

Items can be forced to always appear in the overflow, regardless of the width of the main toolbar, shown on the right of the toolbar by the three vertical dots, and shown dropped here:

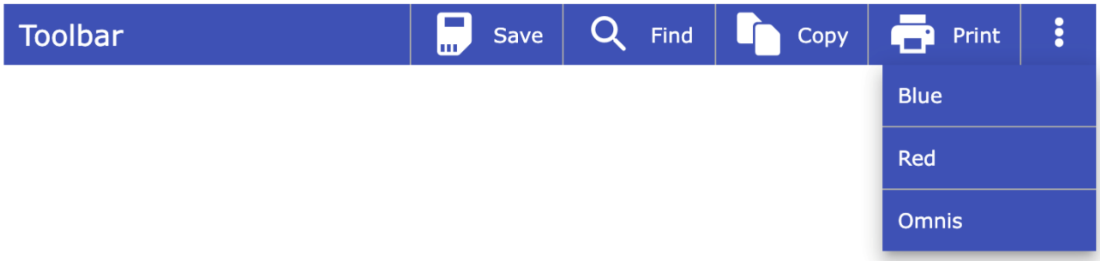


Figure 212:

As the browser window is resized, or the remote form (the app) is displayed on a mobile device, the main toolbar width will shrink, and the button items are added to the overflow menu automatically, as shown (note the icons are not displayed when the item is in the overflow menu):

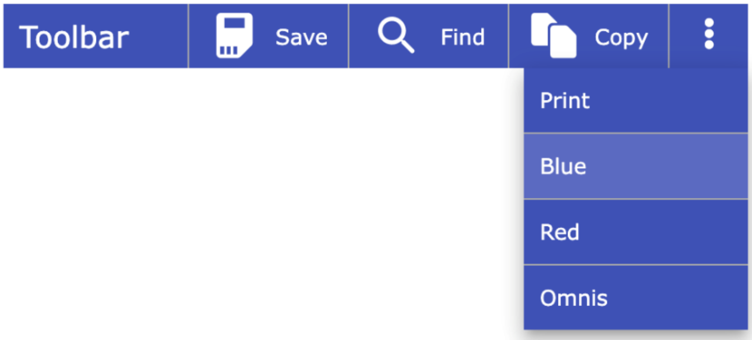


Figure 213:

The following image shows the same Toolbar with the side menu added and in the dropped state:

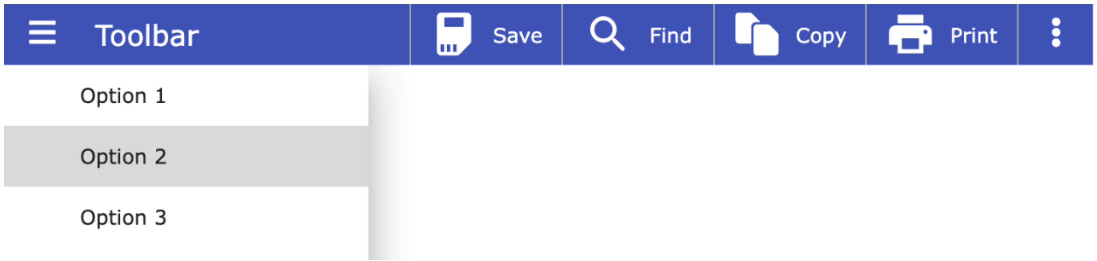


Figure 214:

The following image shows the same Toolbar using the 'lemonade' JS Theme; the material icons used in the toolbar are themed so will change color depending on the theme:

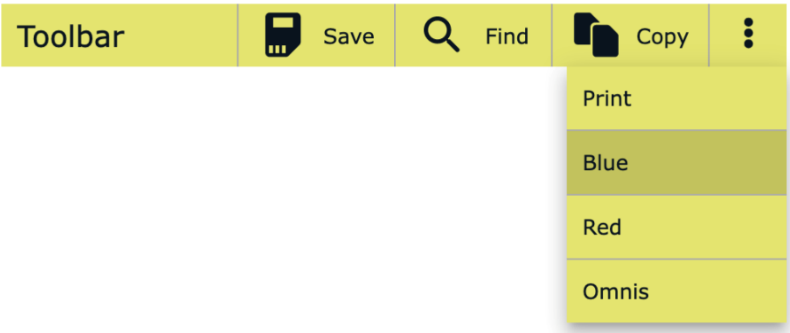


Figure 215:

Properties

The custom properties for the Toolbar control are described below. The 'Item' tab in the Property Manager (shown when the Advanced option is enabled) contains item specific properties that apply to the $currentitem.

Property	Description
$itemcount	The number of toolbar items/buttons
$currentitem	Item specific properties are assigned to the current item
$moveitem	Allows you to move an item in design mode; the current item moves to the position specified by the number entered
$itemiconid	The icon for the current item (item specific property); note $showitemicons needs to be enabled to display icons
$itemtext	The text for current item (item specific property); note $showitemtext needs to be enabled to display text
$itemoverflow	Force the item to be appear in the overflow menu (item specific property)
$itemenabled	If kTrue (the default) the item is enabled (item specific property); if kFalse the item is greyed and cannot be selected with the pointer or keyboard; applies to items whether they are on the toolbar itself or the overflow menu
$sidemenu	Add a side menu to the toolbar. The $dataname must be a list containing the menu data
$dataname	Name of a list variable containing the menu data, to display a menu when $sidemenu is set to true

Property	Description
$verticaltoolbar	If kTrue, the toolbar is in a vertical orientation
$menudirection	A kJSToolbarMenuDirection… constant which sets the direction the menu should open. Different values are available depending on the $verticaltoolbar property: Down or Up for horizontal toolbars, Right or Left for vertical toolbars
$toolbartitle	The optional title to display on the toolbar; leave this blank for no title
$titlefontsize	The font size applied to the toolbar title
$itemwidth	The width of items on the toolbar; without a value items have a variable width and are forced to fit the length of the toolbar
$displaystyle	A kJSToolbarStyle… constant determining the position of the icon text relative to the icon: either Above, Below, Left of, or Right of the icon
$showitemicons	If true, any item with an $itemiconid will display an icon on the toolbar
$showitemtext	If true, any item with $itemtext will display text on the toolbar; when true, $showtooltips is disabled
$showtooltips	Show tooltips on toolbar items; $showitemtext must be set to false
$showdividers	If true, dividers will be shown between toolbar items
$showselecteditem	If true, the item will have its background color set to $selectedcolor and its text colour set to
$selecteditem	The number of the selected item
$clippopuptocontainer	If kTrue (the default), the Side menu and Overflow menu are clipped to the toolbar's container. If false, they can extend outside its bounds

Clicking on a toolbar item will make that item selected, and $selecteditem is set to the selected item number. If $showselecteditem is true, the item will have its background color set to $selectedcolor and its text color set to $selectedtextcolor.

The following properties control the color of toolbar items.

Property	Description
$dividercolor	The color of the dividers between items if $showdividers is true
$iconcolor	The color of standard icons such as the hamburger icon
$sidemenucolor	The background color of the side menu
$overflowcolor	The background color of the overflow menu
$toolbaractivecolor	The color of toolbar items when clicked
$toolbarhovercolor	Hover color for toolbar items
$sidemenuhovercolor	Hover color for side menu items
$overflowhovercolor	Hover color for overflow items
$selectedcolor	The background color of the selected item
$selectedtextcolor	The text color of the selected item
$selectedlinecolor	The color used for the background of the selected line in the side menu
$toolbarhovertextcolor	Text color of toolbar items when hovered
$sidemenutextcolor	Text color of side menu items
$sidemenuhovertextcolor	Text color of side menu items when hovered
$overflowtextcolor	Text color of overflow menu items
$overflowhovertextcolor	Text color of overflow menu items when hovered

Disabling Items

The $itemenabled property allows you to disable specific items. When $itemenabled is set to kFalse for an item it is greyed and cannot be selected with the pointer or keyboard. This property applies to items whether they are on the toolbar itself or the overflow menu.

Defining the Side Menu list

To enable the side menu, you need to set $sidemenu to kTrue and specify a list variable name in $dataname containing the contents of the menu. The $dataname can generate either a grouped or an ungrouped menu.

Ungrouped list columns with each row representing an item:

- Text (Character): The text of the menu item.
- IconPath (Character): A URL of an image to display. The image will be scaled to fit. You can use the iconurl() function to return a URL for an icon in an icon set, e.g. iconurl('apps+32x32') will return the relative URL for the apps icon at 32x32 pixels

Grouped list columns with each row representing a group:

- SubList (List): A list with columns matching the ungrouped list above. Contains data for the group.
- GroupName (Character): The text displayed on the group header.
- Fixed (Boolean): Optional column. If true, the group is always expanded. False by default.
- Collapsed (Boolean): Optional column. If true, the group is collapsed by default. False by default.

Events

The Toolbar reports two events: **evClick** reports the toolbar item that was clicked, with pClickedItem returning the item number; and **evNavigationClick** reports true if an item on the side menu was clicked, with the group number reported in pClickedMenuGroup (zero if the data is ungrouped) and the item number in pClickedMenuItem. You can write event handling code in the $event for the toolbar to trap these events and branch according to the value of pClickedItem or pClickedMenuItem.

Trans Button Control

Group	Icon	Name	Description
Buttons	👆	Trans Button	Interactive button with alternate h image

The **Trans Button** control (or TransButton) is like a standard button, but it can display a different icon and/or background color when the end user's pointer hovers over the control, or when the button is tapped on touch devices. In all other respects the Trans button is like a standard push button control, insofar as it generates a single evClick event when the button is clicked which can be used as confirmation or to initiate an action in your code using the $event method. Note the evClick event must be enabled in the $events property for the control for it to be reported.

There is an example app called **JS Trans Button** in the **Samples** section in the **Hub** in the Studio Browser showing how you can use the Trans button, and the same app is available in the JavaScript Component Gallery.

The following image shows the JS Trans button example app using the 'soft' theme; the material icons used in the buttons are themed so will change color depending on the theme:

Hover Action

The Trans button has several properties prefixed "$hot" that relate to the appearance of the button for the hover action. You can specify two icons for the Trans button: one to represent the "off" state which is specified in $iconid, and the other to represent the "over" state which is specified in $hoticonid: if no $hoticonid is specified the icon in $iconid is used which will not provide a hover effect. You can also specify a different background color for the hover action in $hotbackcolor, and an alternative border color in $hotbordercolor.

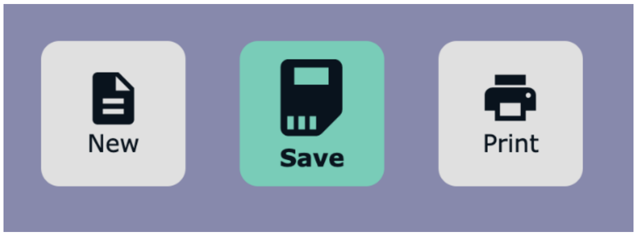


Figure 216:

Icons

The icons used in $iconid and $hoticonid can be from an icon set, or #ICONS, or an icon datafile. You can use the standard icon sizes (16x16, 32x32, 48x48 and their 2x equivalents for hi-def support), but you can also use non-standard sized images as well, but in this case, they can only be sourced from an icon set (and the size should be specified in the file name according to the usual naming). The Trans button will not draw or position icons from standalone pages in #ICONS or icon datafiles (e.g. omnispic.df1) correctly in the client, since the client cannot determine the icon size from its URL.

You can set $vertical to true to center the button's content vertically: $align also affects the placement of the icon when $vertical is true. The $horzpadding property allows you to set the left and right padding.

HTML Button Text

You can specify a label for the button using the $text property which is a single line of plain text. When set to kTrue the $textishtml property specifies that the text in $text is treated as HTML, and therefore any HTML, including character and color attributes, can be used to style the text. For example, you can insert a line break by setting $textishtml to kTrue, and using <br> in $text for the button wherever a line break is required.

The HTML needs to be valid for it to be rendered in the form, including when used as the contents of a <p> element; so for example, you cannot use a <p> element inside another <p> element.

Tree List Control

Group	Icon	Name	Description
Lists		Tree List	List for displaying hierarchical data options

The **Tree List Control** (or Tree Control) provides a graphical way of displaying a list of items in a hierarchical format. Each node can have a check box or its own icon. The $dataname property for a tree list is the name of a list variable containing the content (data) and structure of tree list. The list can contain the entire data for a tree list when the form is opened, or the content can be built as nodes are expanded when a tree list is in "dynamic" mode.

There are two example apps in the **Samples** section in the **Hub** in the Studio Browser showing how you can create a **Tree list** with check boxes (left) and a **Dynamic tree** structure (right); the same apps are available in the JavaScript Component Gallery. The following images show the 'vintage' JS Theme in use.

Properties

Tree lists have the following properties:

Property	Description
$checkbox	If true, and $multipleselect is also true, the tree control has check boxes that can be used to select nodes
$currentnodeident	The current node ident for Dynamic tree lists only: see below

Property	Description
$evenrowcolor	Specifies the color to be used for every even row in the list of nodes. The kColorDefault setting means use the same color as odd numbered rows ($backcolor).
$extraspace	Adds extra space in between the lines in the list; it is the number of pixels added to the normal font height of a row in the list, or zero for no extra space
$iconurlprefix	All icons used in the tree (as a result of $showicons being true) must come from a single icon directory; the default is _icons/omnispic/
$lineborder	If true, a row border is added between each node
$linebordercolor	Specifies the color of the line when $lineborder is true; it uses the value of $bordercolor when set to kColorDefault
$multipleselect	If kTrue allows the end user to select more than one line; should be enabled for tree lists with $checkbox enabled
$nodeaction	Performs an action on the tree node, only for Dynamic mode: see below
$nodedata	List for tree node when building dynamically: see below
$showicons	If true, the tree control shows node icons from location in $iconurlprefix
$showlines	If true, the tree control displays dotted lines connecting nodes
$twostate	If true, and the tree control has checkboxes (see $checkbox), selection of each node is independent

Tree List Format

The format or mode of the data is set in the $datamode property which controls how the list content is used to structure the contents of the tree list. There are several different data modes to format a tree list, as well as the Dynamic mode, represented by the following constants:

- **kJSTreeFlatList**
  The first N columns represent a node in a tree of depth N. The last 5 columns are node properties: iconid,ident(int),expanded(bool),textcolo means $textcolor),tooltip
- **kJSTreeFlatListWithTags**
  The first N columns represent a node in a tree of depth N. The last 7 columns are node properties: iconid,ident(int),expanded(bool),textcolo means $textcolor), tooltip, tag(char), enterable(bool)
- **kJSTreeFlatListOld**
  A list with the same structure as the plug-in client tree kTreeDataFlatList
- **kJSTreeFlatListOldWithTags**
  A list compatible with the plug-in client kTreeDataFlatListWithTags

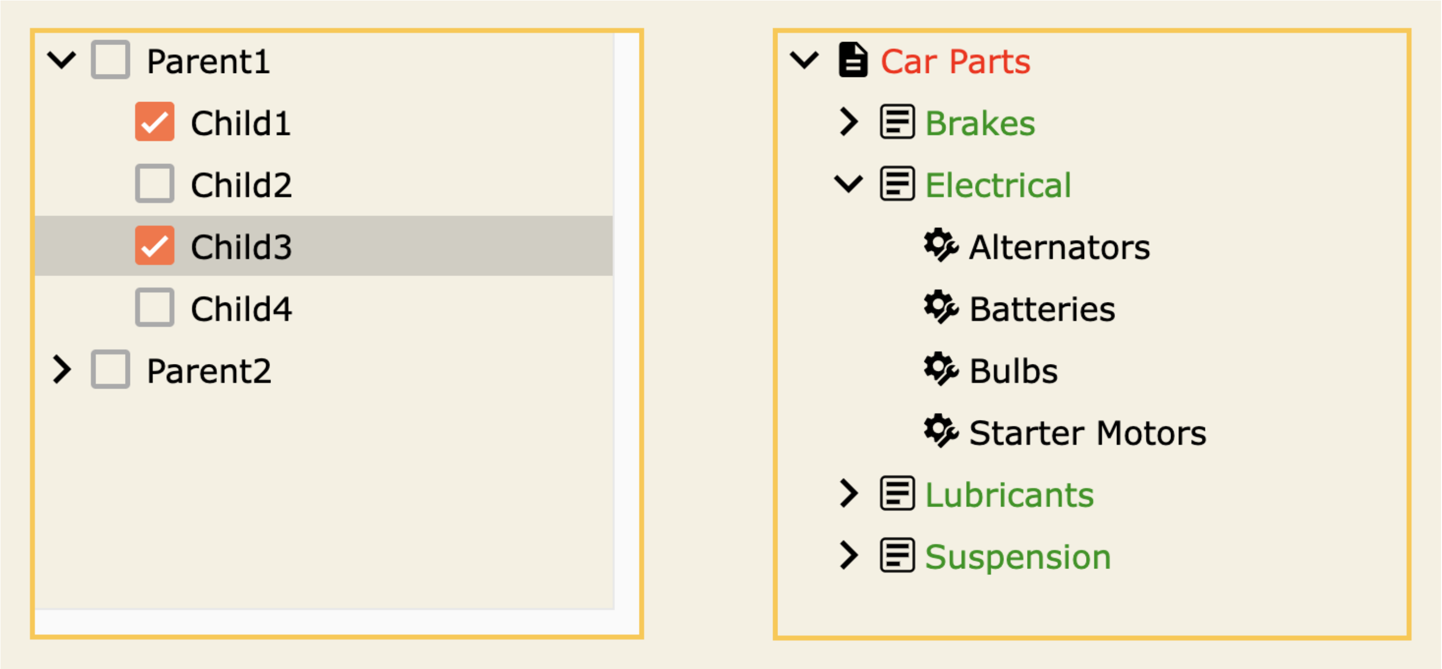


Figure 217:

- **kJSTreeDynamicLoad**
  the tree list content can be built dynamically; see below

Tree Events

When the user selects a node the evClick event is reported, while a double-click reports an evDoubleClick. In both cases, the id and tag of the selected node is reported in the pNodeIdent and pNodeTag parameters. Note you cannot get click or double-click events for nodes which are enterable, as the click puts them into edit mode.

For enterable nodes, the evRenamed event is reported if the end user has renamed the node. evRenamed has node ident, node tag, old name and new name as event parameters.

The evExpandNode event is fired after the user has expanded a node, every time that node is expanded (unlike evLoadNode which is only triggered if the node has no children). This applies to both dynamic and non-dynamic tree lists; see below for dynamic tree lists.

You cannot use $nodedata to load data into the tree list with evExpandNode, it is just a notification and includes the parameters pNodeIdent and pNodeTag. If evLoadNode and evExpandNode are both active, evLoadNode will be fired first, as evExpandNode is fired after the node is expanded.

Scrolling in long lists

Tree lists scroll to view the current line in the list, and any parents opened as necessary, whenever the current line changes or the $currentnodeident property is set. In a non-multiple select tree lists, setting either the current line in the list or the $currentnodeident will select the line and scroll to it. This applies to flat list trees only.

In multiple select tree lists, setting the current line will scroll to that line but will not select it. Setting $currentnodeident notationally will set both the current line and select the node. Tree lists without checkboxes will clear any current selection, but tree lists with checkboxes will not. This behavior mimics the user behaviour of clicking a node.

In multiple select tree lists, the $currentnodeident and the current line in the list can reference different nodes; however, in single select tree lists, they will always reference the same nodes.

Dynamic Tree Lists

The content inside a Tree List Control can be built dynamically as the end user expands a node. In previous versions, the entire contents of the tree list had to be built and sent to the client, including the contents for all unexpanded nodes, which for large lists created quite an overhead. Building the tree list data and displaying content in a tree list can be optimized with the ability to build node contents as required, by setting the tree list data mode to "dynamic"; note there is an example app in the **Samples** section in the **Hub** in the Studio Browser and in the JS Component online gallery.

Creating Dynamic Trees

To use a Tree Control in dynamic mode you need to set its $datamode property to kJSTreeDynamicLoad. Note that dynamic mode can only be used for 'single-select' trees, and attempting to assign kJSTreeDynamicLoad to $datamode will fail if $multipleselect or $checkbox for the tree control is kTrue.

A dynamic tree still requires a list to be identified by $dataname, but the list need only contain the initial content of the tree, that is, the content for the root or parent nodes. After the list content is changed, the tree reloads its content from the list.

Dynamic trees also use lists to set the content of expanded nodes and to add nodes programmatically to the tree. The lists all have the same structure: each list represents an ordered set of nodes with the same parent (or no parent in the case of the $dataname list), and the columns are as follows:

- **Column 1:** Text. The text displayed for the node.
- **Column 2:** Icon. Only used when $showicons is set to kTrue. This is a line number in the list identified by $nodeiconlist, or zero if the node does not have an icon. $nodeiconlist is described below.
- **Column 3:** Ident. A unique positive integer that identifies the node. Cannot be the same as the ident of any other node in the tree. The tree control throws an exception (resulting in a message box displayed on the client) if you try to use a duplicate ident.
- **Column 4:** Tag. A string associated with the node. Any value that is useful to the developer. Need not be unique.
- **Column 5:** Tooltip. The tooltip string for the node, displayed when the user hovers the mouse over the node. Leave empty for no tooltip, although on some browsers nodes may inherit their tooltip from their parent node if the tooltip is empty.
- **Column 6:** Text color. The text color for the node (an integer RGB value). Zero means use the $textcolor of the tree.
- **Column 7:** Flags. Integer flags. A sum of zero or more of the following constants:
- **kJSTreeFlagEnterable**. The node text can be edited (this works with the existing evRenamed event).
- **kJSTreeFlagHasChildren**. The node has children.
- **kJSTreeFlagExpanded**. The node will be immediately expanded. If you set this flag you must supply the child nodes using a node list supplied as the children column.
- **kJSTreeFlagDiscardOnCollapse**. If set, when the user collapses the node, the tree deletes the node contents. This means that the next time the user expands the node, the tree will generate an evLoadNode event; evLoadNode is described later in this document.
- **Column 8:** Children. If the kJSTreeFlagHasChildren is present, you can pre-populate the node content by using a nested list to specify the children. If you do not supply any children using this list, then you can supply them later by using the evLoadNode event (see below). The content of this column is ignored if the kJSTreeFlagHasChildren is not present. You can nest children lists arbitrarily deep (within reason).

$nodeiconlist allows you to specify the icons to be used with the tree. These must be available when the tree is updated from the dataname list. $nodeiconlist must be the name of an instance variable list with at least one column which must be a character column containing icon URLs. You can populate each URL in the node icon list using the iconurl() function, for example:

```
Do iNodeIcons.$define(iNodeIcon)
Do iNodeIcons.$add(iconurl(1710))
Do iNodeIcons.$add(iconurl(1711))
Do iNodeIcons.$add(iconurl(1712))
```

Populating Expanded Nodes

Dynamic trees have the event evLoadNode which allows you to populate the tree on demand, by only populating node content when a node is expanded. When using the kJSTreeDynamicLoad mode for $datamode, evLoadNode is generated so that you can set the content of the node by setting the $nodedata property which is the name of a list containing the expanded node content. For other settings of $datamode, evLoadNode is generated when the user expands a tree node.

The evLoadNode event has two parameters, pNodeIdent and pNodeTag, corresponding to the node that is being expanded. In the event processing for evLoadNode, you can set the $nodedata property to a node list representing the content of the node (with the above column format); if the event processing fails to set this property, the tree control sets the node content to empty.

The list assigned to $nodedata can specify nested children if desired.

$nodedata is a runtime-only property that can only be set.

Manipulating Tree Nodes

Dynamic tree lists support 'node actions' to allow you to manipulate the nodes in a Tree List programmatically. For example, you can expand or collapse a node, and you can add, delete or rename nodes. You execute a node action by assigning a row variable to the $nodeaction property of the tree; this is a runtime-only property that can only be set. The supported actions are as follows:

Node Action	Description
kJSTreeActionExpand	row(kJSTreeActionExpand, ident). Expands the node with the specified ident if it is not already expanded
kJSTreeActionCollapse	row(kJSTreeActionCollapse, ident). Collapses the node with the specified ident if it not already collapsed
kJSTreeActionRename	row(kJSTreeActionRename, ident, newname). Renames the node with the specifie ident to the new name
kJSTreeActionDelete	row(kJSTreeActionDelete, ident). Deletes the node with the specified ident (also recursively deletes node children). If the current node is deleted, an evClick event w be generated to inform the application of the new current node
kJSTreeActionAdd	row(kJSTreeActionAdd, ident, position, nodelist). Adds the nodes in the nodelist to the tree, where ident specifies the parent node of the nodes in nodelist; to add a ne root nodes specify ident as zero. The position parameter can be one of: -1 to add th new nodes before any existing children of the node specified by ident 0 to add the new nodes after any existing children of the node specified by ident a child node ident. New nodes are added after the child node with this ident. If no such node exists, the new nodes are added after any existing children
kJSTreeActionUpdateIcon	row(kJSTreeActionUpdateIcon, ident, newicon) updates the icon of a node. Extra parameter is the line number in the list identified by $nodeiconlist, or zero for no icon
kJSTreeActionMove	row(kJSTreeActionMove, ident, position) moves the node to a new position, where ident specifies the new parent node; to move to the root, specify ident as zero. The position parameter can be one of: -1 to move before any existing children of the noc specified by ident 0 to move after any existing children of the node specified by ide a child node ident. The node is moved after the child node with this ident. If no suc node exists, the node is moved after any existing children.
kJSTreeActionCollapseAndDiscard	row(kJSTreeActionCollapseAndDiscard, ident) collapses the node and discards all child nodes
kJSTreeActionReload	row(kJSTreeActionReload, ident). Reloads the node, generating an evLoadNode event

Collapsing a Node

The tree control generates evCollapseNode when the user collapses a node. This applies to all trees, not just dynamic trees.

The Current Node

The property $currentnodeident applies to dynamic trees; in previous versions the current (selected) node was represented by a list line. When the end user clicks on a node, the value of $currentnodeident changes, and the control generates evClick. In addition, the developer can assign $currentnodeident (and read its value in client-executed methods). $currentnodeident is a runtime-only property.

Video Player

Group	Icon	Name	Description
Media	📹	Video Player	Plays a YouTube or other hosted vi

The **Video Player** control allows you to play a video within your remote form. The video can be hosted on YouTube or you can play a video via the HTML5 video support in the browser. There is an example app called **JS Video** in the **Samples** section in the **Hub** in the Studio Browser showing how you can embed a YouTube video in your remote form; the same app is available in the JavaScript Component Gallery.

The JS Video control has the following properties:

Property	Description
$dataname	If $youtube=kFalse, this is a 2 columned list containing the location and type of the videos to be played: Column 1 is the URL of the video file; Column 2 is the media types If $youtube=kTrue, this is a list containing the IDs of the YouTube videos to be played; only Column 1 of the list is used and contains the ID of the YouTube video.
$showcontrols	If true, the control displays video controls such as the Play and Pause buttons
$youtube	If true, the control will play a movie from youtube.com; column 1 of the $dataname list is the YouTube video id. If false, the control will use HTML5 video to play video files from URLs.
$startposition	The time (in seconds) at which the video should start when played.
$currentposition	(Runtime only) The current time (in seconds) of the current position in the video. Assign to this to ‘seek’ to a particular time.
$duration	(Runtime only) The duration of the current video (in seconds). Read-only (in a client-exec method).
$poster	A URL to an image to display before the first frame of the video is ready. HTML5 video only ($youtube=kFalse).
$playing	Whether the video is currently playing. Assign to this in order to play or pause the video. Note that many mobile devices prevent the playing of videos if not in direct response to a user action.
$volume	The volume level of the video player (0-100). Assigning 0 will mute the player.
$playbackrate	The video’s playback speed, with 1.0 being default speed. Youtube will round down to the closest supported rate of the particular video.

Property	Description
$requestcaptions	If true, closed captions will be turned on (when available, attempting to use the client's language) for Youtube videos. Note that even if disabled, captions may be enabled through the video controls, or through the user's account settings in Youtube (if they are signed in).

Properties relating to the current video player ($currentposition, $duration, $volume) will return -1 if queried before a video is 'ready' (see evVideoReady).

YouTube

If you set the $youtube property to kTrue, the $dataname for the video control should be a list containing YouTube ids: the data in the first column of the first row in the list is used to reference the video on YouTube. Note the YouTube id is not the full URL on youtube.com, but just the id on the end of the URL, e.g. 'Ff-qlTlSkc0'.

Playlists

If the list assigned to a Youtube video control ($youtube = ktrue) has more than one line, a YouTube playlist will be created, using the video IDs supplied in each line of the list. The videos in the playlist will be played successively.

If $showcontrols is true, the playlist can be accessed via the video controls in the UI. You could notationally skip to the next video by skipping to the end of the current video. For example, using the client-executed method:

```
Calculate $cinst.$objs.youtubeVideo.$currentposition as $cinst.$objs.youtubeVideo.$duration
```

HTML5

If you set $youtube to kFalse, the $dataname should be a 2 column list. The first column should contain URLs to the video files, located somewhere on the internet or the Omnis App Server, and the second column should state the media type. For example:

```
Do iList.$define('VideoURL','VideoType')
Do iList.$add('videos/myVideo.mp4','video/mp4')
Do iList.$add('videos/myVideo.ogv','video/ogg')
Do iList.$add('videos/myVideo.webm','video/webm')
```

Not all browsers are able to play all video file types, so you should provide the video in multiple formats and populate the list with the URL to each file and type. When the client connects, the browser will play the first video file it is able to play from the list.

Events

The JS Video control has the following events:

- **evVideoReady**
  Sent when the video is ready to be played, and can be interacted with.
- **evVideoEnded**
  Sent when the video has finished playing (i.e. it has played to the end).

Both events receive a pVideoURL parameter, describing the currently playing video. For HTML5 videos ($youtube = kFalse), this will be a URL to the video file. For Youtube videos ($youtube = kTrue), this will be the Youtube video ID. These should correspond to a value in the list assigned to the control's $dataname.

External Components

iCalendar

The **iCalendar** external component (or Calendar Control) allows you to load and manage calendar events: it is a non-visual External Component that you can use in your Remote Forms (or window classes). iCalendar allows you to read, write and modify objects based on the standard iCalendar format, which is supported by many third-party calendar products.

The iCalendar model is based on four object types:

- **Component:** A group containing Properties which represent, for example, an event. Components can contain other Components (sub-components).
- **Property:** Used to communicate information about a Component, such as a description or a location.
- **Value:** Properties have a value associated with them. For example, a DTSTART Property will have a date or datetime value.
- **Parameter:** A modifier for a Property. Properties may have more than one Parameter (or none).

There are two types of objects in the Omnis iCalendar external component:

- **Document:** Represents the entire Document and its children.
- **Component:** Used to access and manipulate iCalendar Components and their associated Properties and Parameters.

Creating a Calendar Object

To access the iCalendar object and its methods, you need to create an ***instance variable*** in your remote form (or window class), choose Object as its Type, under Subtype drop down the Select object dialog, open the 'iCalendar Objects' group and select 'Document' object.

Working with iCalendar files

iCalendar Documents can be initialized with character data, or built up with the Omnis iCalendar methods.

To load an iCalendar file into a Document object, use FileOps to read in the character data. Then use $initwithdata() to initialise the document.

```
Do FileOps.$getfilename(lPath,"Select ics file","*.ics")
    Do lFileOps.$openfile(lPath,kTrue)
    Do lFileOps.$readcharacter(kUniTypeUTF8,iCalText)
    Do lFileOps.$closefile()
    Do lDoc.$initwithdata(iCalText)
```

To output the character data, use $getdata() on the Document or a single Component. To save the data into a file, use FileOps.

```
Calculate lDocText as iNewDoc.$getdata()
    Do FileOps.$putfilename(lPath,,"*.ics")
    Do lFileOps.$createfile(lPath)
    Do lFileOps.$openfile(lPath)
    Do lFileOps.$writecharacter(kUniTypeUTF8,lDocText)
    Do lFileOps.$closefile()
```

Updating sub-components

The functions $getcomponent() and $getsubcomponent() return a copy of a Component. Therefore, modifying the returned Component will not affect the parent (the Component or Document that the method was called on). In order to update the parent, Component copy will need to be saved back to the parent with $replacerootcomponent() or $replacesubcomponent().

Custom Properties

As well as the standard Property types, custom Properties can be added to Components. These must be prefixed with "X-", e.g. "X-PROPERTY". By default, the data type of a custom Property is character. When adding a Property to a Component with $addproperty(), the optional iDataType Parameter can be used to override the default data type. Doing so will set the "VALUE" Parameter to the data type associated with the constant. The data type cannot be changed after it has been created.

Custom Parameters

Like custom Properties, custom Parameters can be applied to Properties. Similarly, they must also have "X-" as a prefix.

Error Properties

When a Document is initialised with $initwithdata(), the character data is parsed. If there are any syntactic or semantic errors in the data, such as a misspelt Property name or a Property without a value, an X-LIC- error Property will be inserted. For example, the following error is caused by misspelling the ATTENDEE Property:

X-LIC-ERROR;X-LIC-ERRORTYPE=PROPERTY-PARSE-ERROR:Parse error in property name: ATENDEE

Special Values

These values contain multiple parts and are therefore represented as rows. The static $createrow() helper method can be used to build these rows, which can be used to create a new Property or update a value.

Recur

The "RECUR" value type in the iCalendar model denotes a recurring event. It is commonly used with the "RRULE" Property. Its value may contain several parts, separated by semicolons. The parts contain key value pairs separated by the equals sign. The example shows a Recurrence Rule property.

RRULE:FREQ=MONTHLY;BYMONTHDAY=1;UNTIL=19980901T210000Z

A Component's $propertylist will store an RRULE Property as a rows containing columns relating to each keyword. To create an RRULE Property with $addproperty(), the vValue parameter can take either a character or row argument. To create a recurrence type row, use $createrow(kICalendarRowTypeRecur).

Duration

The "DURATION" value type is represented in a component's $propertylist as a row. The columns are: IS_NEGATIVE, DAYS, WEEKS, HOURS, MINUTES and SECONDS. To add a Property with a duration type, a row containing these column names can be used. The column values are all Integers, with the exception of IS_NEGATIVE, which is a Boolean. Alternatively, a string can passed. For example, P15DT5H0M20S denotes a duration of 15 days, 5 hours, and 20 seconds. See the RFC 5545 iCalendar specification for details on this format (https://tools.ietf.org/html/rfc5545#section-3.3.6).

Period

"PERIOD" value types have two parts: The first is the start time (date time). The second can either be the end of the period (date time), or a duration. Period is the default value type of the Free/Busy Property. In the $propertylist, period values are displayed as a row containing a START date time and either a DURATION or an END date time.

19970101T180000Z/19970102T070000Z date time "/" date time (Explicit)

19970101T180000Z/PT5H30M date time "/" duration (Start)

Geo

"GEO" Properties hold geographic coordinates, represented as two floats separated by a semicolon. The values are latitude and longitude respectively, e.g. 37.386013;-122.082932.

In the $propertylist of a Component, they are displayed as a row containing a LAT and a LONG column with float values.

Static Methods

$createcomponent()

OmnisICalendar.$createcomponent(iType)

Creates a new iCalendar Component object using one of the kICalendarComponent... constants. Returns an iCalendar component object.

- iType: A kICalendarComponent... constant to specify the Component type.

$createrow()

OmnisICalendar.$createrow(iType)

Returns a row which can be used to add or update certain Properties. iType can be one of the kICalendarRowType... constants.

- iType: A kICalendarRowType... constant to specify the type of row.

Document Object Methods

$initwithdata()

$initwithdata(cData)

Initializes the object with the Character contents of an iCalendar file. Returns true if successful.

- cData: The character data containing the contents of an iCalendar file.

$getdata()

$getdata() - no parameters

Returns character data representing the Document that can be saved as an iCalendar file.

$getcomponent()

$getcomponent(iComponentId)

Returns a copy of the root Component object with the specified ID.

- iComponentId: The ID of the root Component to find.

$addrootcomponent()

$addrootcomponent(oComponent)

Adds a Component to the root of the Document. Returns the ID of the new Component.

- oComponent: The Component to be added to the root.

$deleterootcomponent()

$deleterootcomponent(iComponentId)

Removes the Component with the specified ID from the root. Returns true if the Component was deleted.

- iComponentId: The ID of the Component to delete.

$replacerootcomponent()

$replacerootcomponent(iComponentId, oComponent)

Replaces the Component at the specified ID with oComponent. Returns true if successful.

- iComponentId: The ID of the Component to replace.
- oComponent: The new Component.

Document Object Properties

$componentlist

A list of root-level Component info for the Document. The columns for this list are: ID, Type and TypeName.

Component Object Methods

$getdata()

$getdata() - no parameters

Returns character data representing the Component and its children, that can be saved as an iCalendar file.

$isvalidcalendar()

$isvalidcalendar() - no parameters

Returns true if the VCALENDAR Component meets the RFC 5546 iCalendar specification standards.

$getsubcomponent()

$getsubcomponent(iComponentId)

Returns a copy of the sub-component object with the specified ID.

- iComponentId: The ID of the sub-component to find.

$addsubcomponent()

$addsubcomponent(oComponent)

Adds a sub-component to the Component. Returns the ID of the new Component.

- oComponent: The sub-component to be added to the Component.

$deletesubcomponent()

$deletesubcomponent(iComponentId)

Removes the sub-component with the specified ID from the component. Returns true if the sub-component was deleted.

- iComponentId: The ID of the Component to delete.

$replacesubcomponent()

$replacesubcomponent(iComponentId, oComponent)

Replaces the Component at the specified ID with oComponent. Returns true if successful.

- iComponentId: The ID of the Component to replace.
- oComponent: The new Component.

$addproperty()

$addproperty(cName, vValue, [wParameters, iDataType])

Adds a new Property to the Component. Returns the Property ID.

- cName: The name of the Property. Must be a valid Property type.
- vValue: The value to assign to the Property. The type can be Character, Integer, Date Time, Float, Boolean or Row.
- wParameters: A row of Parameters to add to the Property.
- iDataType: A kICalendarDataType... constant. Sets the 'VALUE' Parameter which overrides the default data type for the Property. Can be used to specify the type of a custom Property.

$deleteproperty()

$deleteproperty(iPropertyId)

Delete the Property with the specified ID. Returns true if the Property was deleted.

- iPropertyId: The ID of the Property to delete.

$setparameter()

$setparameter(iPropertyId, cName, cValue)

Sets the Parameter of the Property. If there is an existing Parameter with the same name, it will be overwritten. Returns true if successful.

- iPropertyId: The ID of the Property associated with the Parameter.
- cName: The name of the Parameter to set.
- cValue: The value to set the Parameter to.

$updateproperty()

$updateproperty(iPropertyId, vValue, [wParameters])

Updates the Property with the specified ID. Providing Parameters will overwrite any existing ones. Returns true if successful.

- iPropertyId: The ID of the Property to update.
- vValue: The new value to assign to the Property. The type can be Character, Integer, Date Time, Float, Boolean or Row.
- wParameters: A row of Parameters to add to the Property.

$deleteparameter()

$deleteparameter(iPropertyId, cName)

Removes the Parameter with the name cParamName from the Property. Returns true if the Parameter was deleted.

- iPropertyId: The ID of the Property associated with the Parameter.
- cName: The name of the Parameter to delete.

Component Object Properties

$componentlist

A list of sub-component info for the Component. The columns for this list are: ID, Type and TypeName.

$propertylist

A list of iCalendar Properties held by this Component. The columns for this list are: ID, PropertyName and PropertyValue. The PropertyValue column contains a row for each Property. Each row has a "_VALUE" column containing the Property value (the type of this depends on the Property), and columns for any Parameters that the Property has.

$typename

The type name of the Component.

$typenumber

The type number of the Component.

Chapter 4—JSON Components

You can define your own remote form controls using JSON and JavaScript, and use them on JavaScript Remote forms in your web and mobile applications. Using the same technique, you can wrap ready-made JavaScript components, available from any third-party, opening up endless possibilities for new controls to use in your web and mobile apps.

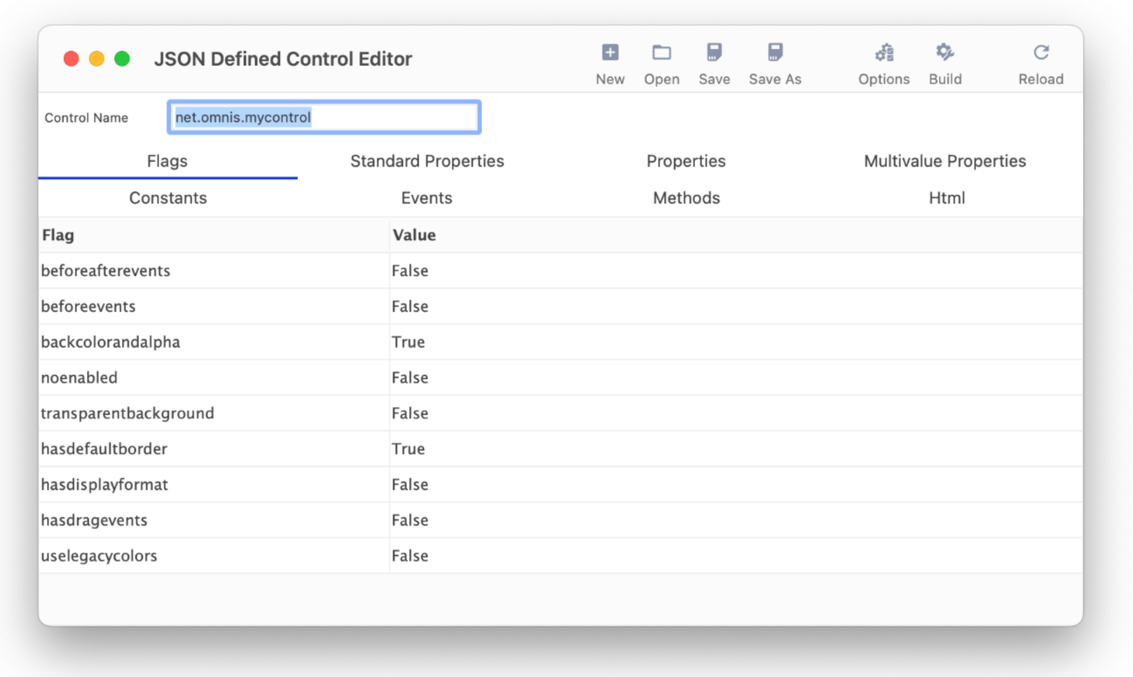
This method of creating JavaScript components provides an alternative to creating external components using C++ and our JavaScript SDK, which is the current method used for creating JavaScript external components. It also means you only need to understand JSON and JavaScript, together with our JavaScript interfaces on the client, in order to create and use JSON defined JavaScript controls, either in your own web or mobile apps, or to provide to the wider Omnis community. There are a number of JSON components on our GitHub site: https://github.com/OmnisStudio, for example, Omnis-Signature (for signing documents) and Omnis-FiveStars (star rating component).

Having created a JSON defined component using the JSON Control Editor, the component will appear in the Component Store in the **JSON Components** group. You can drag the component onto your JavaScript remote form and set its properties using the Property Manager.

The design mode rendering of the JSON controls on a remote form is very basic, and does not reflect the actual control as it might appear on a remote form at runtime, although this is not a problem for some controls that do not require a visual interface.

JSON Control Editor

A JSON control is defined in a JSON file, called a **JSON Control Definition** (JCD) file, which you can create or edit using any text or JSON editor – if you are very familiar with JSON you may like to create the JCD using an editor. Alternatively, you can create the new JSON controls (create a JCD file) using a tool available under the Tools>>Add Ons menu, called the **JSON Control Editor.**



Control Name: net.omnis.mycontrol

Flag	Value
beforeafterevents	False
beforeevents	False
backcolorandalpha	True
noenabled	False
transparentbackground	False
hasdefaultborder	True
hasdisplayformat	False
hasdragevents	False
uselegacycolors	False

Figure 218:

The JSON Control Editor contains a template control that has all the necessary properties to create a basic JSON control. The editor allows you to set the properties for the control under each tab. To create a component, you edit the properties, click on Save, click on Build to build the control, and then click on Reload to load the component into the Component Store (the Build and Reload options will also prompt you to save if changes have been made). The New button removes any changes you have made to the default template and allows you to start again. In order to setup the properties and methods for your control you will need to refer to the JSON definitions later in this section.

Control Name

The name of the control must be unique, so you will need to change the **Control Name** in the editor (or just accept the default name if you are testing the editor). The default control name is prefixed with 'net.omnis' to show the preferred naming convention, but you should change this to your own company name, e.g. com.mycompany.mycontrol1, or use any appropriate naming convention. If you do use a dot in the control name, Omnis converts it to underscore, since dots cause an issue with the Omnis notation.

Control Properties

The following tabs are available to set the properties of the control:

- **Flags**
  allows you to set whether or not events are enabled, whether or not the control has a transparent background, whether or not drag events are enabled, and so on
- **Standard properties**
  an array of standard properties supported by the control, in addition to the basic properties such as name
- **Properties**
  an object defining the control-specific properties of the control; the name of each member of the properties object is the name of the control property, without the leading $, e.g. id, type, etc.
- **Multivalue properties**
  allows you to set up a control to have multiple values for certain properties
- **Constants**
  an object defining the constants for the control, e.g. value, desc, etc.
- **Events**
  defines the events that the control generates (in addition to those specified by the flags member) and including the standard events such as evClick; the name includes the “ev” prefix
- **Methods**
  specifies the client-executed methods that the control provides; the method name includes the “$” prefix
- **Html**
  specifies how the initial HTML sent to the client for the control is generated

The ‘Options’ item on the editor toolbar allows you to set custom JavaScript variable prefix for properties.

The Save option places the JSON control file in html/controls folder. The Build option places the JavaScript file for your control in the html/scripts folder in your development tree. It also prompts you to include a reference to the JavaScript file for the control in the **jsctempl.htm** file, which will ensure that the control is available for testing any remote forms that contain the new control. The Build option adds update markers and lets you update the JavaScript file if the markers already exist.

When you have built a JSON control you need to restart Omnis for it to load. After restarting Omnis, the control will appear under the **JSON Components** tab in the Component Store ready to use in your remote forms. When you deploy your app, you need to place the JSON and JavaScript files in the corresponding folders in your Web Server tree, and check that they are referenced in the html page containing your remote form.

You could open the ‘control.json’ file created in the JSON Control Editor when you build the control from the template: this file will show you the typical structure of the JSON file required to define a new component.

Using Ready-made JS Components

When using ready-made JS components, that you have obtained from a third-party, you need to add the .js file(s) to the html/scripts folder in the Omnis tree, and any other CSS and image files required for the control need to be put in the appropriate folder(s). You will also need to add any properties, methods, and events in your JS control to the JSON definition file via the JSON Control Editor. There is a tech note on the Omnis website that describes the process of using a ready-made JS component in Omnis:

- TNJC0009: Adding Ready-made JavaScript components to Omnis

You will also need to refer to the JavaScript Control Reference in the **JavaScript Component SDK** docs which you can find here.

- TNJC0015: You may like to read this Tech note about using a JSON Component to add Signature Capture to a Remote form

JSON Control Definition

This section describes the different properties that can be defined in the JCD file for the control and edited under the separate tabs in the JSON Control Editor (or when editing separate members using a text editor).

There is a new folder in the Omnis tree, html/controls, which contains a sub-folder for each JSON control you have defined. The names of these sub-folders are not critical, but it makes sense to use the same name as the control name.

The JSON Control Editor will create html/controls folder when you build your first control, otherwise if you are building your own controls you will need to create this folder (note this is not to be confused with the 'htmlcontrols' folder that contains controls that can be loaded in the oBrowser object).

There is a new external component named 'jsControls' in the jscomp folder, which handles all JSON defined controls. It loads and validates the controls at startup. All controls which pass validation are loaded into the new JSON Components group in the Component Store. If a control fails validation, jsControls opens the trace log, and adds a message to indicate there is a problem with the control. The exact problem can be found in a file called control_errors.txt in the control's folder.

Each control must have a unique name. This is defined in control.json (see below), and you should use a convention similar to Java except that Omnis uses underscore rather than a dot, e.g. net_omnis_control1 could be the name of a control (using dots causes issues in the Omnis notation).

JSON Control Object

Each JSON control folder must contain a file named **control.json** containing a JSON object defining the control; the JSON file *must be called control.json*. The members of this object are defined in the following sections.

The control folder should also contain an image file which is used for the control icon in the Component Store, and when rendering the control on the remote form design window. This can be an SVG file (with the .svg extension) which can be themed. Alternatively, you can use PNG files (with the .png extension), but in this case, you need to provide all possible sizes including 16x16, 16x16_2x, 48x48 and 48x48_2x.

name

The name member is mandatory, and it specifies the name of the control; it becomes the external component control name. It is also used to create the JavaScript control class name, as ctrl_<name>. For example:

```
  "name": "net_omnis_control1"
```

In this case the JavaScript control class would be ctrl_net_omnis_control1.

flags

The flags member is mandatory. It is an object that allows certain features of the control to be configured. Each member of flags is optional, and defaults to false if it is omitted. Valid members are:

- **beforeafterevents** and **beforeevents** (are mutually exclusive)
  indicate if the control supports either evAfter and evBefore, or just evBefore respectively. If both are omitted, the control supports neither event (see also the events member)
- **backcolorandalpha**
  indicates if the control has backcolor and backalpha properties.
- **noenabled**
  indicates if the control does not have the enabled property.
- **transparentbackground**
  indicates that the control has a transparent background, and does not have backcolor and backalpha properties. Must not be used with backcolorandalpha set to true.
- **hasdefaultborder**
  indicates if $effect for the control can have the value kJSborderDefault.

- **hasdisplayformat**
  indicates if the control has date and number format properties.

- **hasdragevents**
  indicates if the control has drag events (see also the events member).

- **uselegacycolors**
  a Boolean, it is automatically set to True when loading existing JSON controls so the existing colors are used. The flag defaults to False for all new controls which means they can use theme colors

For example:

```
"flags": {
    "beforeafterevents": true,
    "backcolorandalpha": true,
    "noenabled": true,
    "hasdefaultborder": false,
    "hasdisplayformat": true,
    "hasdragevents": true
},
```

standardproperties

The standardproperties member is optional. It is an array of standard properties supported by the control; inclusion in the standardproperties member means the control will have the property. These are over and above the basic properties that apply to all controls e.g. active, name, etc.

Valid members of the standardproperties array are: "dataname", "effect", "bordercolor", "borderradius", "linestyle", "font", "textcolor", "align", "fontstyle", "fontsize", "horzscroll", "vertscroll", "autoscroll", "dragmode".

For example:

```
"standardproperties": [
    "dataname",
    "effect",
    "bordercolor",
    "borderradius",
    "linestyle",
],
```

properties

The properties member is mandatory. It is an object defining the control-specific properties of the control. Each member of the properties object is itself an object that contains members that describe the property. The name of each member of the properties object is the name of the control property, without the leading $. Valid members of each property object are:

- **id**
  The identifier of the property. A positive integer. This is mandatory, and it is a critical field in that Omnis stores this value in the copy of the object saved in the class, in order to identify the property. This means you must not change id values after you start to use the control on a remote form. id must be unique for all properties for the control. When jsControls loads the control, it will validate property id uniqueness. It usually makes sense to start numbering your properties at 1.

- **desc**
  The description of the property. A character string. This is mandatory, and is used by the IDE, for example, as the property tooltip in the Property Manager. Double quote and backslash characters are escaped when saving desc items

- **tab**
  An optional member. A character string that identifies the Property Manager tab to be used for the property. Defaults to the Custom tab if omitted. Otherwise, it must have one of the following values: custom, general, data, appearance, action, prefs, text, pane, sections, java or column.

- **type**
  A mandatory member. A character string that identifies the type of the property. This can be one of the basic types (number, integer, character, boolean or list) or a specific type (color, dataname, font, icon, pattern, fontstyle, linestyle, multiline, set, or remotemenu).

- **runtimeonly**
  An optional member. A boolean which is true to indicate that the property is a runtime only property. Defaults to false if omitted.

- **findandreplace**
  An optional member. A boolean which is true to indicate that the property is searched by find and replace. Defaults to false if omitted.

- **extconstant** or **intconstant**
  Optional members. A boolean which is true to indicate that the property is constrained to a range of constants defined by this control (extconstant), or by the Omnis core (intconstant). They are shown in the Property Manager. They can be used for both integer type properties, and set type properties; in the latter case, the first member of the range must be a constant that has the value zero, and represents the empty set. You must define either extconstant or intconstant, so they cannot both be set to true.

- **constrangestart** and **constrangeend**
  These members must be present if either extconstant or intconstant is true. In the case of intconstant, they are integer constant idents that specify the range of constants - you can see the idents for core constants in the $constants group in the notation inspector. In the case of extconstant, these are the names of constants defined by this control; the members of the range are the constants starting with constrangestart, and ending with constrangeend, in the order they occur in control.json. Note that when used with a set, the constant values need to correspond to the bit mask used to represent the set. If intconstant is selected, a constant name such as kPlain entered into constrangestart or constrangeend is converted to its ident value.

- **min** and **max**
  These members are optional, and only apply when the type is integer. They specify minimum and maximum values for the property.

- **initial**
  This member is optional. It can be used to specify an initial value for the property. For number types, it can be a floating point number. For character types, it is a character string: double quote and backslash characters are escaped when saving. For integer types, it is an integer. For boolean types it is a Boolean: values of ‘true’ or ‘kTrue’ are overridden with 1. The initial value is used, for example, when dragging a new copy of the control out of the Component Store (provided that a copy of the control is not already stored in the Component Store).

- **editreadonly**
  An optional member. A boolean which is true to indicate that the property is a read-only property. Defaults to false if omitted.

For example:

```
"properties": {
   "headercolor": {
       "id": 1,
       "desc": "The header color of the control",
       "type": "color",
       "tab": "appearance",
       "initial": 255
   },
   "headericon": {
       "id": 2,
       "desc": "The header icon of the control",
       "popuptype": "icon",
       "tab": "appearance"
   },
   "rangeofexternalconstants": {
       "id": 3,
       "desc": "Range of external constants",
       "type": "integer",
       "extconstant": true,
            "constrangestart": "kNetOmnisControl1Range1",
            "constrangeend": "kNetOmnisControl1Range3",
        }
    }
```

multivalueproperties

The multivalueproperties member is optional. It allows you to set up a control to have multiple values for certain properties. It is an object with members as follows:

- **itemlistproperty**
  This is mandatory. When a control supports properties with multiple values, the properties are stored in a list. Each row of the list contains the set of properties for a particular tab or column. We call the tab or column (or something else) an item. This property must have type list, and it is automatically hidden from the property manager.
- **itemcountproperty**
  This is mandatory. It is the name of an integer property defined by the properties member, that can be set to specify the number of items in the item list. You can specify a max value for this property in order to restrict the number of items, otherwise it is restricted to no more than 256 items.
- **currentitemproperty**
  This is mandatory. It is the name of an integer property defined by the properties member, that identifies the current item displayed in the property manager, and to which property changes apply to multi-value properties.
- **moveitemproperty**
  This is mandatory. It is the name of an integer property defined by the properties member, that can be used to move the current item to a new position in the item list.
- **properties**
  This is mandatory. It is an object that specifies the properties that have multiple values, and where they are stored in the list. Each member must be the name of a property in the main properties object; the value of each member is the list column in the item list where the property value is stored. It is important not to change the column number once you have started using the control.

For example:

```
     "multivalueproperties": {
        "currentitemproperty": "curitem",
        "itemlistproperty": "itemlist",
        "moveitemproperty": "move",
        "itemcountproperty": "itemcount",
        "properties": {
            "mvprop1": 1,
            "mvprop2": 2
            }
        }
    }
```

constants

The constants member is mandatory. It is an object defining the constants for the control. Each member of the constants object is itself an object that contains members that describe the constant. The name of each member of the constants object is the name of the constant. Valid members of each constant object are:

- **value**
  The value of the constant. An integer. This is a mandatory member.
- **desc**
  The description of the constant. A character string. This is mandatory, and is used by the IDE for example as the tooltip in the catalog.

- **group**
  The catalog group to which the constant belongs. This is optional. By default, all constants defined for the control belong to the group “RF:jsControls-<control name>”. You can use this member to replace the control name with something else. All constants occurring after the constant with the group specified belong to this group, until a new group is specified (if any).

For example:

```
"constants": {
    "kNetOmnisControlHeaderColor": {
        "value": 123,
        "desc": "The description of this constant"
    },
    "kNetOmnisControl1Range1": {
        "value": 3,
        "desc": "Range constant 1",
           "group": "Ranges"
    },
    "kNetOmnisControl1Range2": {
        "value": 5,
        "desc": "Range constant 2"
    }
}
```

events

The events member is optional. It specifies the events that the control generates (in addition to those specified by the flags member, i.e. before, after, and drag events). Each member of the events object identifies an event. The name of each member is the name of the event, including the “ev” prefix. Certain standard events can be specified: evClick, evDoubleClick, evTabSelected, evCellChanges, evHeaderClick and evHeadedListDisplayOrderChanged. Valid members of each event object are:

- **id**
  Must not be specified for standard events. Otherwise, this is mandatory, and is the positive integer id of the event. This id must match the event id you use in the JavaScript implementation of the control, and must be unique within the context of this control.
- **desc**
  Must not be specified for standard events. Otherwise, this is mandatory, and is a text string describing the event.
- **parameters**
  The event parameters of the event. This is an array. Each array member is an object with members as follows:
  name
  This member is mandatory. The parameter name. Do not include the p character prefix - Omnis will add this. Note that if you use re-use an event parameter name, then the remaining members of this object are ignored, and overridden by the original definition of the parameter - the first control (or Omnis core) using a name sets the type and description of that parameter.
  type
  This member is mandatory. The data type of the parameter. integer, character, boolean or list.
  desc
  This member is mandatory. A text string describing the parameter.

For example:

```
"events": {
     "evNetOmnisControlOpened": {
         "id": 1,
         "desc": "The event sent when the control opens",
         "parameters": [
             {
                 "name": "name",
                 "type": "character",
                 "desc": "The name event parameter"
             },
             {
                 "name": "name2",
                 "type": "integer",
                 "desc": "The second event parameter"
             }
         ]
     },
     "evClick": {
         "parameters": [
             {
                 "name": "zname1",
                 "type": "character",
                 "desc": "The zname1 event parameter"
             },
             {
                 "name": "zname2",
                 "type": "integer",
                 "desc": "The zname2 event parameter"
             },
             {
                 "name": "horzcell",
                 "type": "character",
                 "desc": "the horz cell event parameter"
             }
         ]
     }
```

methods

The methods member is optional. It specifies the client-executed methods that the control provides. Each member of the methods object identifies a method. The name of each member is the name of the method, including the “$” prefix. Valid members of each method object are:

- **id**
  This is mandatory, and is the positive integer id of the method. It must be unique within the context of this control. It is used internally by the Omnis core.
- **desc**
  This is mandatory, and is a text string describing the method.
- **type**
  This is mandatory. The return type of the method. integer, boolean, character or list.
- **parameters**
  This member is optional. It is an array describing the parameters of the method. Each member of the array is an object with the following members:
  name
  This is mandatory. The name of the parameter. Omnis will insert a data type character at the start of this name.
  desc
  This is mandatory. A text string describing the parameter.
  type
  This is mandatory. The data type of the parameter. integer, boolean, character or list.
  altered
  Optional. A boolean, default false. If true, the parameter is marked as one that will be altered.
  optional
  Optional. A boolean, default false. If true, the parameter is marked as optional.

For example:

```
"methods": {
    "$mymethod1": {
        "id": 1,
        "desc": "This is my method",
        "type": "integer",
        "parameters": [
            {
                "name": "p1",
                "type": "character",
                "altered": true,
                "desc": "The parameter p1"
            },
            {
                "name": "p2",
                "type": "integer",
                "desc": "The parameter p2",
                "optional":true
            }
        ]
    }
}
```

html

The html member is mandatory. It specifies how the initial HTML sent to the client for the control is generated. It is an object with members as follows:

- **template**
  Mandatory. A character string that is a template for the inner div of the control. For example: <div %o %s data-props='%p' data-mvprops='%m'></div>:
  jsControls replaces **%o** with the JavaScript client attributes for the client element, which includes the id attribute of the client element: this element must be specified.
  jsControls replaces **%s** with the style attribute for the div, based on the normal Omnis processing and the properties the control supports.
  jsControls replaces **%p** with the control properties that are not multi-value. %p is replaced with a JSON string, representing an object, where each member of the object is named by the property name, with value of the property value. The value may have been mapped by Omnis to what the client will require, for certain property types such as color and icon. The client JavaScript can use this string to create an object containing its property settings.
  jsControls replaces **%m** with the multi-value control properties. %m should be omitted if the control does not use such properties. %m is replaced with a JSON string, similar to %p, except that it is an array of objects, with an array entry for each multi-value item.
- **extrastyles**
  Optional. A string of length up to 255 characters of extra style attributes to include in the style attribute replacing %s in the template, e.g. "margin:2px".
- **padding**
  Optional. An integer used to set padding (in pixels) in the style attribute replacing %s.
- **relativeposition**
  Optional. Boolean, default false. If true, the style attribute replacing %s includes position relative rather than absolute.
- **nowrap**
  Optional. Boolean, default false. If true, the style attribute replacing %s includes white-space nowrap.

For example:

```
"html": {
    "template": "<div %o %s data-props='%p' data-mvprops='%m'></div>",
    "extrastyles":"margin:1px;"
}
```

The resulting inner div for the control looks like this:

```
<div style='position:absolute; top:0px; left:0px; height:106px; width:88px; font- family:'Times New Roman',Geo
```

Note that it is important to use single quotes around the attributes in the template, since JSON includes double quotes. jsControls escapes any single quotes in the JSON it inserts into the place-holders as \u0027.

customtabname

The customtabname member is optional. If specified, it is the name of the custom properties tab for the control shown in the Property Manager.

JavaScript

When you have created a JSON control and added it to your Omnis tree, you can add the supporting JavaScript file to the remote form template in the HTML folder (the JSON Control Editor will do this automatically). To do this, you can add:

```
    <script type="text/javascript" src="scripts/ctl_net_omnis_mycontrol.js"></script>
```

to the scripts section of the jsctempl.htm file (in the html folder) so the control is always included in the test HTML page for your remote form; it also needs to be included in the HTML page serving your deployed web or mobile app.

Chapter 5—Ultra-thin Omnis

In addition to accessing the Omnis App Server via the JavaScript Client, Omnis Studio lets you interact with a Remote Task in your Omnis application using standard HTML forms. This approach is often referred to as “Ultra-thin Omnis” since the client’s browser uses standard GET and POST methods without any other client access layers.

In the Ultra-thin Omnis environment, the HTML form sends parameters and data directly to the Omnis App Server via the Omnis web server plug-in located in the cgi-bin or scripts folder. When the ‘Submit’ button in your HTML form is pressed, the Omnis web server plug-in is executed and passed all the form’s parameters. The Omnis web server plug-in sends the request to the Omnis App Server, which creates an instance of the specified Remote Task Class and calls its $construct method.

HTML Forms and Remote Tasks

The parameters required in the GET request in your HTML form contain the necessary criteria for interacting with the Omnis Remote Task in the Omnis library running on the Omnis App Server. Your HTML contains the location of the web server plug-in, the port number, the remote task name, the Omnis library name, as well as the specification for the fields in the form. For example, the following constructs a form with the GET method:

```
<form method="GET" action="/cgi-bin/omnisapi.dll">
  <input type="hidden" name="OmnisServer" value="PortNumber">
  <input type="hidden" name="OmnisClass" value="RemoteTaskName">
  <input type="hidden" name="OmnisLibrary" value="LibraryName">
  <p><input type="password" name="pPassword" size="20"></p>
  <p><input type="text" name="pQuery" size="80"></p>
  <p><input type="submit" value="Submit" name="B1">
  <input type="reset" value="Reset" name="B2"></p>
</form>
```

The form has the special parameters:

- **OmnisServer**
  specifies the port number or service name of the Omnis App Server, that is, the value specified in the $serverport preference (in the range 1-65535) in the Omnis App Server.
- **OmnisClass**
  the name of the remote task class to which this form is to connect (not a remote form as is the case with JavaScript Client based applications).

3. **OmnisLibrary**
   the internal name of the library containing the remote task class on the Omnis App Server; the default is the library file name minus the .lbs extension

The following example HTML source code implements a feedback form. The Omnis specific parameters are marked in bold; the remainder of the source specifies the form fields and text labels in the form, including a standard Submit button.

```
<form method="GET" action="/cgi-bin/omnisapi.dll">
 <input type="hidden" name="OmnisClass" value="tFeedback"> ;; the remote task name
 <input type="hidden" name="OmnisLibrary" value="FeedbackApp"> ;; the library name
 <input type="hidden" name="OmnisServer" value="5912"> ;; the port number
 <table border="0" cellspacing="0" cellpadding="0" width="760">
  <tr>
   <td width="788" valign="top"><div align="right">
    <p><strong><font face="Arial">Developer Name:</font></strong></p></div></td>
   <td width="564" height="25"><font face="Arial">
    <input type="text" name="Name" size="27"></font></td>
  </tr>
  <tr>
   <td width="788"><div align="right">
    <p><strong><font face="Arial">Serial No:</font></strong></p></div></td>
   <td width="564" height="25"><font face="Arial">
   <input type="text" name="Serial" size="27"></font></td>
  </tr>
  <tr>
   <td width="788" valign="top"><strong><font face="Arial">
    <div align="right">
    <p>Platform:</font></strong></td>
   <td width="564" height="23"><table border="0" cellspacing="0" cellpadding="0" width="520">
    <tr>
      <td width="135"><font face="Arial">
      <input type="checkbox" name="Macintosh" value="YES">   <strong>Macintosh</strong></font></td>
      <td width="385"><font face="Arial">
      <strong><input type="checkbox" name="Windows" value="YES">Windows</strong></font></td>
    </tr>
  </table>
 </td>
</tr>
<tr>
 <td width="788" valign="top"><strong><font face="Arial">
  <div align="right"><p>Client:</font></strong></td>
 <td width="564" height="23">
  <table border="0" cellspacing="0" cellpadding="0" width="601">
   <tr>
    <td width="136" height="13"><font face="Arial"> <input type="checkbox" name="ActiveX" value="YES"> <st
    <td width="135" height="13"><font face="Arial"><strong><input type="checkbox" name="Netscape" value="Y
    <td width="330" height="13"><font face="Arial"><strong> <input type="checkbox" name="RAWHTML" value="Y
   </tr>
  </table>
 </td>
 </tr>
 <tr>
```

```
     <td width="788" valign="top"><strong><font face="Arial"> <div align="right"> <p>Comments:</font></strong>
    <td width="564" height="249" valign="top"> <font face="Arial"><textarea rows="11" name="Comments"
cols="57"></textarea></font></td>
  </tr>
  </table>
  <div align="center"><center><p><font face="Arial">
  <input type="submit" value="Send Comments" name="B1"></font></p>   </center></div>
</form>
```

In the above example, the HTML form uses the "GET" method. You can also use the "POST" method. The main difference lies in how the data is transmitted to the server, and this is reflected in cases where the URL generated by the form is displayed in the location bar of the user's browser. A "GET" method is equivalent to requesting the URL

```
/cgi-bin/omnisapi.dll?OmnisClass=…&OmnisLibrary=…&…
```

whereas a "POST" method is equivalent to posting to the URL

```
/cgi-bin/omnisapi.dll
```

and sending content which specifies the parameters. If the URL is being displayed in the location bar, you might choose to use "POST" in order to hide the parameters from the user.

Secure Sockets

You can use secure sockets (HTTPS) if you have installed an SSL certificate on your web server. Omnis will use a secure connection to connect the client to the web server if you prefix the location of your HTML forms with "https://". In addition, remote tasks have the $issecure property that lets you turn secure mode on and off dynamically, by assigning to the property for the current task at runtime.

Using Task Methods to Process Requests

Omnis passes a single parameter to the $construct() method of a remote task instance created by a request from an HTML form. This parameter called pParams is a row variable, and it has a column for each parameter in the HTML form.

The code contained in the $construct() method of your remote task could literally do anything you like and depends on the functionality of your application. Having passed in the row variable to your $construct() method you can use any type of Omnis programming to process the information submitted by the user and pass back whatever content or data you wish. Typically and with most complex web applications your library will contain many different methods that can be called from your remote tasks, most of which could be located in object classes to allow reuse of your code. See the *Omnis Programming* manual for information about programming Omnis methods and using object classes.

The column names in the row variable are the same as the form parameter names, and the values are either the values of the parameters in the form (e.g. hidden fields may have default values), or the values entered by the user. You can use the columns values in the pParams row variable (i.e. values in the form parameters) as input to what you do in $construct().

The following $construct() method is contained in a remote task that handles Studio evaluation download form requests. The method contains a parameter variable called pParams of Row type that receives the values from the form. The row variable contains a column for each field in the form, including the fields Email, Country, Platform, and so on, that are referenced using the syntax pParams.Colname. The method also contains several calls to methods in object classes within the library handling the download requests. After registering the customer, the method sends the customer an email and redirects their browser to the appropriate download page.

```
# $construct() for processing download requests
If len(pParams.Code)
  Do iObj.$getMagazine(pParams.Code,lMagazineRow) Returns lCodeFound
# get magazine data (esp. name and serial number)
Else
  Calculate lCodeFound as kTrue
# default to true for straight downloads
End If
If not(lCodeFound) ;; invalid magazine code
  Do inherited
  If not(len(pParams.Folder)>0)
```

```
    Do method createFolderName Returns lFolder
    ; identify the folder name for this user
  Else
    Calculate lFolder as pParams.Folder
  End If
  Do iHTMLObj.$setFolder(lFolder)
  # set the folder name for this instance of the html page
  Do iHTMLObj.$setUrl(iUrl)
  Do iHTMLObj.$createRegisterError(pParams,lFolder)
Else
  Do iObj.$getDownloadCustID(pParams.Email,lCustID)
  # see if this is an existing download customer
  Do lAdminObj.$getDistID(pParams.Country) Returns lDistID
  If not(lCustID)
    Do iSequence.$getNext('ECS_DownloadCustomers') Returns lNextSeq
    # get the next sequence number
    Do iObj.$createDownloadCustRecord(lNextSeq,pParams,lDistID)
    Calculate lCustID as lNextSeq
  Else
    Do iObj.$getDownloadCustData(lCustID,lCustRow)
    # get their data
    Calculate lCustRow.DistID as lDistID
    # if the user has changed country their DistID may
    # also have changed, so update it anyway
    Do iObj.$updateDownloadCustData(pParams,lCustRow)
    # update their account data
  End If
  Do iObj.$getPlatformID(pParams.Platform,lPlatformID)
  # get the platform
  Do iSequence.$getNext('ECS_Downloads') Returns lNextSeq
  # get the next sequence number
  Do iObj.$createDownloadRecord(lNextSeq,lCustID,lPlatformID,,,,pParams.Code,       pParams.ProductID,pParams.H
  If len(pParams.Code)
    # if they have entered a magazine code, they need an
    # email with a serial number
    # send user email with the appropriate serial number
    Do iEmailObj.$sendEmail(2,lCustID,lMagazineRow.SerialNumber,,kTrue)
    # 4th parm is blank, 5th parm for using downloadcustomer table
    Calculate lUrl as 'http://www.omnis.net'
  Else
    Do iEmailObj.$sendEmail(1,lCustID,,,kTrue)
    # 3rd & 4th parm are blank; 5th parm is for using
    # downloadcustomer table
    Switch pParams.Platform
      Case 'Win95'
        Calculate lUrl as iWin95download
        Break to end of switch
      Case 'WinNT'
        Calculate lUrl as iWinNTdownload
        Break to end of switch
      Case 'Ppc'
        Calculate lUrl as iPpcdownload
        Break to end of switch
      Case 'Linux'
        Calculate lUrl as iLinuxdownload
        Break to end of switch
      Case 'OSX'
        Calculate lUrl as iMacosxdownload
        Break to end of switch
      Default
```

```
        Calculate lUrl as iWin95download
        Break to end of switch
    End Switch
 End If
End If
If not(lCodeFound)
  Do $itasks.DOWNLOAD.$getUnsecuredUrl(lServerUrl)
  # get the server path
  Calculate lUrl as con(lServerUrl,sys(9),'downloadhtml',sys(9),lFolder,sys(9),iUrl,'.htm')
End If

Quit method lUrl ;; return the path to the approp download
```

The final command in the $construct() method is the Quit method <url> which returns an URL to the client's browser. See the next section for further details.

Returning Content to the Client

After completing its processing, $construct() returns its results as the return value of the method, using the *Quit method* command. There are three possible types of return from such a remote task instance: <url>, <data>, and <error>.

Quit method <url>

$construct() can generate a file (typically HTML), and any files it references, and then return the URL of that file. For example, it can use the HTML report destination, and print a report to HTML.

You return the <url> in one of two forms.

- If you prefix it with http:// or https://, the user's browser will be redirected to the specified URL. For example,

```
  Quit method "http://www.myhost.com/myfile.html"
```

- If you do not prefix the URL as above, the user's browser will be redirected to the URL http://<web server address><url>. For example, if your web server is www.myhost.com

```
Quit method "/omnishtml/00000001/myfile.html"
```

will result in the browser being redirected to

```
http://www.myhost.com/omnishtml/00000001/myfile.html
```

If you do generate dynamic HTML files, perhaps by printing reports to HTML, you can use the oDirClean object to periodically check for expired files, and delete them. The HTML report task wizard contains an example of how to do this. You should also note that if you are generating new output for each request, you need to use a different file name or folder for each request. The oDirClean object provides a mechanism which allows this.

Quit method <data>

$construct() can generate content directly in memory, and return that content to the user. The content can be of any content type supported by a browser, typically HTML, or perhaps a JPEG image. You build the content in either an Omnis character or binary variable. The content must start with the text "content-type:" to differentiates the data from a <url> or <error> (the other possible returns from $construct). The syntax of the data in the variable is:

- HTTP headers starting with "content-type:" and separated by carriage return-linefeed pairs. There must also be a content-length header.
- An empty line terminated by a carriage return-linefeed pair.

- The content, for example HTML or JPEG data, the length of which must match that specified by the content-length header.

For example:

```
Begin text block
Text: Content-type: text/html (Carriage return,Linefeed)
Text: Content-len 12 (Carriage return,Linefeed)
Text: (Carriage return,Linefeed)
Text: Some con
End text block
Get text block iReturn

Quit method iReturn
```

returns the 12 character string “Some content” to the browser.

If you want to return binary content, you can build up the HTTP headers in a character variable, and then use the *bytecon()* function to combine the character variable and the binary content, storing the result in a binary variable. You can return the result as the return value of *Quit method*.

In addition to the content type and length headers, you can specify other HTTP headers if you wish. However, note that Omnis will automatically generate Expires: and Pragma: no-cache headers, so you should not generate these.

The following extract of code is an HTML based storefront that presents with many different HTML forms for logging in, choosing products, and so on, and heavily uses remote tasks to process user requests and compose HTML on-the-fly to display in the client’s browser.

```
# $create_rtOrder1 method for composing the store login screen
# do some preparation...
Begin text block
Text: <html> (Platform newline)
Text: (Platform newline)
Text: <head> (Platform newline)
Text: <title>Omnis Store - Login</title> (Platform newline)
Do method getCSSHeader
Do method getJSHelpHeader
Do method getHeader (kTrue) ;; kTrue - wide table
Text: <td width="262" ><span class="titletext">Login to the Omnis Store</span></td> (Platform newline)
Text: <td width="220"></td> (Platform newline)
Text: </tr> (Platform newline)
Text: <tr> (Platform newline)
Text: <td width="500" colspan="3"><span class="bodytext"> (Platform newline)
Text: The Omnis Store lets you purchase the Omnis Studio rapid application (Platform newline)
Text: development environment. Ordering is easy, just follow the step by step process.</span></td> (Platform n
Text: </tr> (Platform newline)
Text: <tr> (Platform newline)
Text: <td width="500" colspan="3">   </td> (Platform newline)
Text: </tr> (Platform newline)
Text: <tr> (Platform newline)
Text: <td width="500" colspan="3"><table border="0" width="100%" cellspacing="0" cellpadding="0"> (Platform ne
Text: <tr> (Platform newline)
Text: <td width="500" colspan="3"><form method="Get" action="[iOmnisDll]"> (Platform newline)
Text: <input type="hidden" nam="OmnisLibrary" value="[iLibName]"> (Platform newline)
Text: <input type="hidden" nam="OmnisClass" value="rtNewCustomer"> (Platform newline)
Text: <input type="hidden" nam="OmnisServer" value="[iServerPort]"> (Platform newline)
Text: <input type="hidden" nam="IEBrowser" value="[pParams.IEBrowser]"> (Platform newline)
# store browser type
Text: <input type="hidden" nam="Folder" value="[pParams.Folder]"> (Platform newline)
Text: <input type="hidden" nam="BasketID" value="[pParams.BasketID]"> (Platform newline)
# store basket id if there is one
Text: <input type="hidden" nam="AcctsCountry" value="[pParams.AcctsCountry]"> (Platform newline)
```

```
# so we know the accounts country
Text: <input type="hidden" nam="InitialCountry" value="[pParams.InitialCountry]"> (Platform newline)
# so we know the initla country
Text: <input type="hidden" nam="SessionID" value="[pParams.SessionID]"> (Platform newline)
# so we know the session
Text: <table border="0" width="100%" cellspacing="0" cellpadding="0" bordercolor="#FFFFFF"> (Platform newline)
Text: <tr><td width="50%"> (Platform newline)
Text: <span class="subtitle">New Customers</span><br> (Platform newline)
Text: <span class="bodytext">Click here to create a new account:</span><br> (Platform newline)
Text: <input type=image src="../../images/buttons/newaccount/new1.gif" width="135" border="0" height="39" nam=
Text: onmouseout="MM_swapImgRestore()" onmouseover="MM_swapImage('NewAccount','','../../images/buttons/newacco
Text: </td></tr></table> (Platform newline)
Text: </form> (Platform newline)
# etc etc etc...
```

The above code gives you some idea of how you can use the Text: command to build up complete HTML files and pass them back to the client. Note you can use Omnis variables enclosed in square brackets to fill in parameter values for form objects, such as:

```
<input type="hidden" name="OmnisServer" value="[iServerPort]">
```

Quit method <error>

$construct() can return an error to the browser. To do this, you use *Quit method 'nnn The Error message'*, where *nnn* is a three digit error code followed by a single space before the error text.

Turning off the pragma:nocache header

The "x-omnis-ctrl:" control header can be returned at the start of the content to allow you to turn off the pragma:nocache header.

When returning content, it can either start with "content-type:" or "x-omnis-ctrl:". If x-omnis-ctrl: is present, it must be at the start of the content, to have any effect. To prevent the Omnis web server plug-in from generating the pragma:nocache header, start the content with:

```
x-omnis-ctrl:nopragma<cr><lf>
```

The letters can be in any case, but there must be no white space.

Cookie and Referer Headers

When a request arrives at the web server from an HTML form, there are some HTTP headers sent from the client that may be of interest to the remote task instance. These include "Cookie:" and "Referer". (Note that a discussion of how the various HTTP headers work is beyond the scope of this document; there are many good sources of information on the Web). You can gain access to these parameters using the following mechanism.

In your HTML form, include an empty parameter with the name HTTP_<normalized header name>, where <normalized header name> is the header name in upper case with any – (hyphen) characters changed to _ (underscore), for example, HTTP_COOKIE. This is an instruction to the Omnis web server plug-in, telling it to insert the value of the corresponding HTTP header into the form parameter. This means that when $construct() runs, there is a column HTTP_<header name> in the row variable parameter, and its value is the value of the HTTP header. Note that the header value can be empty if it is either not present, or not available to the Omnis web server plug-in.

Persistent Remote Tasks

Normally, the remote task instance created to process an HTML form destructs, as soon as Omnis has called $construct(), and received the result to return to the user. It is possible that you may want to keep the remote task instance available, so that it can receive further requests. This mechanism is explained here. You should note:

- This mechanism does not work in conjunction with Load Sharing (Load Sharing is described in a later chapter).

- This mechanism does not detect use of the ‘back’ button on the browser, meaning that remote task instances can build up; these only go away when they time-out.

Each time Omnis receives a request from an HTML form, its default behavior is to destruct the remote task instance after returning the result. You can prevent this behavior, by implementing the $canclose method for the remote task instance, and returning kFalse if the task is to stay open. The remote task instance has a property $connectionid, which identifies the particular Omnis web client connection. The data you return to the user must contain $connectionid, so that the next request to the Omnis App Server addresses the remote task instance. Typically, you return a new HTML form to the user. This must contain a parameter “ConnectionID” which contains the value of $connectionid.

evPost events

When an HTML form request arrives at the Omnis App Server, Omnis looks for the ConnectionID parameter. If there is none, processing proceeds in the usual manner, calling $construct() for a new remote task instance. If there is a ConnectionID parameter, Omnis tries to locate the existing remote task instance. If it is no longer present, Omnis returns an error to the user. Otherwise, Omnis sends an evPost event to the $event() method of the remote task instance.

evPost works in exactly the same way as $construct(). In other words, there is a parameter which is a row variable containing the HTML form parameters, HTTP header processing occurs for the parameters, and the result is one of the three alternatives allowed for the result of $construct(). The only difference is that the parameters and return value are handled using event parameters. Event parameter pPostData is a row variable containing the form parameters from the client, and you use the field reference event parameter pPostResult to return the result to the client.

After each call to evPost, Omnis calls $canclose, to see if the remote task instance can now be closed.

Multipart Form Data

You can pass data and upload files from HTML forms to the Omnis App Server, using the Multipart form data type. The content type “multipart/form-data” can be used for submitting forms that contain files, non-ASCII data, and binary data files.

To use multipart form data, add the following attribute to the form tag:

```
enctype="multipart/form-data"
```

and use the file=“type” attribute to identify files to be uploaded. For example:

```
<FORM method="POST" action="/omnis_apache" enctype="multipart/form-data" accept-charset="utf-8">
  <input type="hidden" name="OmnisServer" value="3012">
  <input type="hidden" name="OmnisLibrary" value="LIB">
  <input type="hidden" name="OmnisClass" value="rt">
  <input type="text" name="test">
  <input type="file" filename="myfilename" name="fileparam">
  <input type="submit" value="Send">
</FORM>
```

Note that the “get” method restricts form data set values to ASCII characters. Only the “post” method (with enctype=“multipart/form-data”) will cover the entire character set.

The form parameters are passed to the remote task using a row variable. Parameters without the filename attribute behave as in previous versions, that is, their contents is passed to the row variable. Parameters with the filename attribute are passed to the remote task via a column in the row variable, called MultiPartFileList. This column is a list, with a row for each filename parameter. The list has seven columns:

Column	Description
ParamName	the name of the filename parameter
ReceivedFileName	the pathname of the file as it was specified on the client machine
DataLength	the length of the file data in bytes
IsText	if the length of the file is less than or equal to 16384 bytes, the data is in one of the following two cols, depending on the value of IsText
CharData	if IsText is kTrue, this contains the character data read from the file.

Column	Description
BinData	if IsText is kFalse, this contains the binary data read from the file.
TempFilePath	if the length of the file is greater than 16384 bytes, TempFilePath contains the pathname of a temporary file containing the file data; the temporary file is deleted after the remote task has returned from its $construct()

Direct Client Connections

In addition to the technique of connecting web based clients to an Omnis App Server via a standalone Web Server, you can connect directly to Omnis, without the need to install a Web Server (Omnis has its own built-in web server). Using the built-in http server may be a convenient way to test an Omnis Ultra-thin application, where the only difference is the format of the http call itself to Omnis: in all other respects your application, including the code in your remote tasks, is the same.

To enable direct connections to the Omnis App Server you need to make some modifications to the html file.

- The WebServer script attribute needs to be set to /webclient
- The WebServer url attribute needs to be set to http://<ipaddress>:port

In this case, the OmnisServer address attribute is not relevant when connecting directly to Omnis in this way.

The direct http call to Omnis is structured like this:

```
http://<Server>:<Serverport>/Ultra
```

<Server> is the domain name or the IP address of the computer on which Omnis Studio is running. This is often 127.0.0.1 for your own local machine, but will be configured using a different address or server name on a remote server.

<Serverport> is the port number which has been set in $serverport of Omnis Studio; this is 5912 by default, but you can change it to anything you wish, in the range 1-65535. The $serverport property is an Omnis preference ($root.$prefs).

To try this out, create a library called “DirectHTTP” and add a remote task called “rtDirectHTTP”. Then insert the following code into the $construct() method of the remote task:

```
# $construct() method
# create the following variables
# Parameter var: pParams (Row)
# Local vars: fullhtml (Char 100000000) & html (Char 100000000)
Begin text block
Text: <html> (Carriage return,Linefeed)
Text: <body bgcolor="FFFFFF"> (Carriage return,Linefeed)
Text: <title>Hello [pParams.User] </title> (Carriage return,Linefeed)
Text: <H1>Hello [pParams.User] </H1> (Carriage return,Linefeed)
Text: <a href="javascript:history.go(-1);">Go back</a> (Carriage return,Linefeed)
Text: </BODY> (Carriage return,Linefeed)
Text: </html> (Carriage return,Linefeed)
End text block
Get text block html
Calculate fullhtml as con('content-type: text/html',chr(13,10),'content-length: ',len(html),chr(13,10,13,10),h

Quit method fullhtml
```

The Remote task can be called within an HTML form with the following source text:

```
<html>
  <form action="http://127.0.0.1:5912/ultra" method="Get"> What is your name?
    <input type="Text" name="User" size="30" maxlength="50"></br></br>
    <input type="Submit" name="Send" value="Send">
    <input type="hidden" name="OmnisLibrary" value="DirectHTTP">
    <input type="hidden" name="OmnisClass" value="rtDirectHTTP">
  </form>
</html>
```

You must change the IP address and the port according to your configuration, although the IP address and port number given above are the default values.

Then open the HTML form in a browser, enter a name in the field, and click Send.

This will call the $construct method in the remote task "rtDirectHTTP", passing a parameter called "User" to it, which will contain the text you entered in the form. The full http call is like this:

```
http://127.0.0.1:5912/ultra?User=Username&Send=Send&OmnisLibrary=DirectHTTP&OmnisClass=rtDirectHTTP
```

The remote task processes the form values (and performs whatever other functions you like), and returns standard HTML text that is displayed in the browser.

Chapter 6—Localization

If you are developing web or mobile applications for an international market, using the JavaScript Client, you may want to translate the text and labels in your apps into another language, or support multiple languages. You can use the **String Table Editor** in Omnis to create an external file to store alternative strings for the labels and text in your app to support multiple languages in your JavaScript remote forms. The external file is stored in Tab Separated Value (TSV) format.

Using TSV files is now the preferred method for localization and the String Table Editor uses this format by default. The old way of storing string tables internally (in.stb files) and handling them via $stringtabledata and $stringtabledesignform still works (for Omnis Studio 6.0 or above), but is only present for backwards compatibility and should not be used for new applications. This technique is described in the Localization chapter in the Omnis Programming *manual.*

Localization for the JavaScript Client

String Table Format

The String Table external component uses a Tab Separated Value (TSV) file for storing string tables. TSV files should be stored in the same folder as the library file, and have the following format and features:

- Data is stored in a UTF-8 encoded text file, as a series of rows of fields.
- Fields in the data are separated by tabs.
- Each field is enclosed in double quotes.
- Double quotes in field values are escaped as a pair of double quotes.
- Field can contain newline characters.
- Each row is separated by a newline outside the contents of a field.
- The fields in row 1 are the column names for the string table.

If you open the Catalog (press F9) while editing a remote form or remote task that has a string table associated with it (via $stringtable), then the Catalog automatically loads the string table specified in $stringtable, if it is not already loaded, or reloads it if it has changed on disk.

Localizing Remote Forms

Remote tasks have a property, $stringtable which is the name of the string table for the current library and shared by all JavaScript Client remote form instances in the remote task. When a client connects and the remote task is using the $stringtable property, Omnis loads the appropriate string table.

When you test a remote form that has an associated string table, Omnis generates a JavaScript file automatically if the file does not exist or if the string table file (.tsv) is more up to date than the script file. In addition, Omnis inserts a script tag for the string table into the HTML file generated automatically by the Test Form option. If you change the string table file name in $stringtable, you need to test the form again in order to regenerate the HTML file containing the correct name. Note that only the Omnis development version rebuilds the JavaScript string table file: this is not produced automatically in the Omnis Server.

The path of the string table JavaScript file is of the form:

```
html/strings/libname/file.js
```

where 'strings' is a folder in the html folder, 'libname' is a folder for the library, and file.js is the JavaScript file for the remote task strings, named using the name of the task string table file.

For deployment, you need to place the file.js in the equivalent folder in the web server tree where the other Omnis HTML pages, scripts, etc are located. Alternatively, you can use an option in the String Table Editor to export the string table JavaScript file, rather than using the exported file from your development tree. This option can be used to output the entire table as a JavaScript file, or you can output one or more files for selected locales, where each file contains a single selected locale column and is named file.locale.js.

String tables are converted to separate JavaScript files and transferred to the client as needed and depending on its locale. The script file is cached in the client browser and only reloaded when the string table has changed.

Optimizing string tables

Single-locale JavaScript string tables can be used to further improve loading performance for string tables. There is a new file, jsStringTableSwitch.htm in the html folder in the main Omnis development tree. This file can be used as the initial remote form for an application, and has markers where it can be customized - this allows you to specify the string table file to use for each locale, and a default for unknown locales. In addition, jsStringTableTempl.htm needs to be customized to set up the initial remote form etc, and the string table path. When a page based on jsStringTableSwitch.htm loads, the page:

- Runs a script which selects the string table file to use, based on the locale.
- Loads the template based on jsStringTableTempl.htm using AJAX.
- Sets the string table to use in the template
- Replaces the document content with the modified template

This results in an HTML page for the remote form that only loads the strings for the current locale, and which still has the original URL you have chosen for your application.

Standalone Client

If you run your application in the Standalone client you have to set the library preference $serverlessclientstringtable to specify which string table to use for the remote form instances in the SCAF for the library. This takes the name of a string table (tab-separated value .tsv file in library folder).

String table functions

The client accesses the current string table for the stgettext() function, and properties assigned with the $st prefix. In addition, server methods can also use stgettext() to look up strings for the client locale (either the locale received from the client, or the locale set using $stringtablelocale). If the locale is not present in the string table, stgettext() will return values from column 2 of the string table.

Setting the Client Locale

You can use the 'setlocale' client command to override the locale on the client device in the JavaScript Client.

- **"setlocale"** takes a row with 2 parameters
  cLocale - a string containing valid locale code (e.g. "fr", "en", etc)
  [cPromptToReload] - defaults to false (no dialog opens), if true pops up a no/yes dialog asking the user if they wish to reload the page for language changes to take affect. Defaults to No to reduce the chance of the user accidentally selecting yes and losing any unsaved data.
- **"clearlocale"** takes a row with 1 optional parameter
  [cPromptToReload] - as above

The application needs to be restarted on the client for the change in locale to have an effect. Note that the client commands set this as a localStorage preference, so all Omnis JS client applications (forms) on the same client device will use this setting.

Localizing Built-in Strings

There are a number of strings that can appear in error messages and other dialogs in the JavaScript Client. These are built into the JS client and are in English by default, but from Studio 10.2 onwards, support for German, French, Italian and Spanish translations is provided, while support for other languages can be added as required.

Localized String files

The 'locale' folder under html/scripts/ contains a number of ,js files containing the strings for English and other languages. The files are named in the format strings_[language code].js using either a 2 or 4 letter language code; for example, strings_en.js for standard English, or strings_en_us.js for American English. Inside each file is an object, which is a member of jOmnisStrings, containing key-value pairs to translate to the given language. The member name should match the language code given in the file name, therefore, for french, the strings_fr.js file contains an object jOmnisStrings.fr. The strings_base.js file contains the base strings, and should always be present.

There is a template file called strings_template.js, which provides more information about creating your own translations, with comments for each key-value to help you understand what each string is used for.

Setting the supported languages

The global variable supportedLanguages is defined in the htm file for a remote form (i.e. the application). This contains an array of language codes for the supported languages. On loading a remote form, the locale of the client is detected, and then checked against the supported languages in the array. It will look for both the 4 and 2 letter language codes, and if found, will request strings_base.js and the relevant localized versions. This means the JS client will only download the strings it needs.

Built-in Strings

The following strings are present in the jOmnisStrings strings object in the JavaScript Client, where \x01 is a place-holder which is replaced by parameters added to the string when it is called by the client; you should retain \x01 in your translated text. You can use the contents of strings_template.js for a definitive list of strings.

(Note that some of the error messages are very unlikely to appear in the final deployed version of your app, since they relate to design mode only, so it is not entirely necessary to translate all of the error strings.)

String object	String text (English default)
comms_error	An error has occurred when communicating with the server. Press OK to retry the request
comms_timeout	The server has not responded. Press OK to continue waiting
ctl_dgrd_id	You cannot use \x01 as a list column name for a list bound to a data grid
ctl_dgrd_other	(1 other)
ctl_dgrd_others	(\x01 others)
ctl_file_batchsizeerror	Total batch of files is larger than maximum allowed upload size (\x01)
ctl_file_batchsizetext	\x01 of \x01

String object	String text (English default)
ctl_file_downloaderror	Download error
ctl_file_filesizeerror	File size is larger than maximum allowed upload size (\x01)// \x01 //
ctl_file_filesizetext	\x01 of \x01
ctl_file_filesuploaded	\x01/\x01 files
ctl_file_stopbutton	Cancel upload
ctl_file_uploadbutton	Upload
ctl_file_uploaderror	Upload error
ctl_file_uploadmultipletitle	Upload files
ctl_file_uploadstopped	Upload stopped
ctl_file_uploadtitle	Upload file
ctl_subf_params	Control \x01: $parameters cannot be assigned at runtime
ctl_tree_badgnl	Control \x01: Internal error calling get node line for dynamic tree
ctl_tree_badident	Control \x01: You cannot use \x01 as a tree node ident - tree node idents must be a non-zero positive integer
ctl_tree_dupident	Control \x01: The tree already has a node with ident \x01 - tree node idents must be unique
ctl_tree_invmode	Control \x01: Invalid data mode for tree
disconnected	You have been disconnected. Refresh or restart application to reconnect
error	Error
local_storage_unavailable_error	Unable to access localStorage (perhaps cookies are disabled?).//The application will not run.
omn_cli_badobj	object $objs.\x01 does not exist
omn_cli_callprivate	callprivate cannot call ”\x01”://Exception: \x01
omn_cli_cgcanassign	cannot use $canassign for row section object ”\x01” in complex grid because it has exceptions
omn_form_addbadpage	Parent page number \x01 not valid for paged pane ”\x01
omn_form_addbadparent	Parent object ”\x01” for add control is not a paged pane
omn_form_addcg	Cannot add control to parent object \x01 contained in complex grid
omn_form_addparent	Cannot find parent object \x01 for add control
omn_form_addsrc	Cannot find source object \x01 for add control
omn_form_ctrlinst	Failed to install the control \x01. Possible missing class script
omn_form_nofile	There is no file with the specified ident (\x01)
omn_form_noinstvar	Instance variable does not exist (\x01)
omn_form_readfileerror	Error \x01 occurred when reading the file with ident \x01
omn_inst_assignpdf	Assign PDF: HTML control ”\x01” not found
omn_inst_badformlist	\x01: Invalid formlist
omn_inst_badparent	\x01: Invalid parent for subform set
omn_inst_badpn	\x01: Paged pane does not have page \x01
omn_inst_badpp	\x01: Cannot find the paged pane with name ”\x01
omn_inst_badservmethcall	Cannot make server method call when waiting for a response from the server
omn_inst_badsfsname	\x01: Invalid or empty name for subform set
omn_inst_cliexcep	Exception occurred when executing client method://
omn_inst_dupsfsname	\x01: A subform set with this name already exists
omn_inst_dupuid	Subform set already contains unique id \x01
omn_inst_excep	Exception occurred when processing server response://
omn_inst_excepfile	File ”\x01” Line \x01//
omn_inst_formloaderr	Failed to load form data for \x01. Server returned \x01
omn_inst_formnum	Invalid form number. Parameter error \x01
omn_inst_objnum	Invalid object number. Parameter error \x01
omn_inst_respbad	Unknown response received from server
omn_inst_senderr	Failed to send message to server
omn_inst_sfsnotthere	\x01: A subform set with name ”\x01” does not exist
omn_inst_xmlhttp	Failed to initialize XMLHttpRequest
omn_list_badaddcols	The argument count for $addcols must be a multiple of 4
omn_list_badrow	Invalid list row
omnis_badhtmlesc	Invalid HTML escape
omnis_badstyleesc	Invalid style escape sequence

String object	String text (English default)
omnis_convbad	Error setting \x01: variable type \x01 not supported by JavaScript client
omnis_convbool	Error setting \x01: data cannot be converted to Boolean
omnis_convchar	Error setting \x01: data cannot be converted to Character
omnis_convdate	Error setting \x01: data cannot be converted to Date
omnis_convint	Error setting \x01: data cannot be converted to Integer
omnis_convlist	Error setting \x01: data cannot be converted to List
omnis_convnum	Error setting \x01: data cannot be converted to Number
omnis_convrow	Error setting \x01: data cannot be converted to Row
omnis_escnotsupp	Text escape not supported by JavaScript client
switch_off	OFF
switch_on	ON

Chapter 7—Deploying your Web & Mobile Apps

To deploy your Omnis web or mobile application you need to host it on the *Omnis App Server* which is the main engine at the heart of your Omnis app deployment. You will also need a *Web Server* to host the HTML page(s) containing your remote form(s) and any other web pages or content. When the end user runs your application, the app connects to your Omnis library running on the Omnis App Server, via the web server, and the JavaScript remote forms you have designed are loaded in the end user's desktop browser or the browser on their mobile device. You can deploy your JavaScript Client based application to the web or mobile devices in two ways:

- **Web and mobile app deployment**
  you can deploy your application to the web and provide your end users with a URL to the location of the app, which they can navigate to on their desktop, tablet or mobile device; the initial remote form is embedded into an HTML file which is created for you during development which you will need to edit for deployment

- **Standalone mobile app deployment**
  alternatively you can compile your app into one of *Application Wrappers* to provide a single standalone app for deployment to mobile devices; in effect the wrapper points to the HTML page containing the initial remote form for your app hosted on the Omnis App Server. Download the Application Wrappers.
  A standalone app can operate "online" with a permanent connection to the internet and the Omnis App Server, or it can work "offline" and then reconnect to the Omnis App Server to synchronize data and content. See Creating Standalone Mobile Apps
  Alternatively, a standalone app can operate entirely in "serverless" mode without any connection to the Omnis App Server. See Serverless Client
  On macOS only, you can use the Omnis App Manager to test your standalone app on iOS phones and tablets before uploading them to the Apple AppStore. See Omnis App Manager

In addition to hosting your HTML page, you need to install the Omnis *Web Server plug-in* into your web server which handles all communication between the Omnis App Server and any connected web and mobile clients.

Server Installation and Licensing

You can download the Omnis App Server installer from the Omnis website at: www.omnis.net/developers/resources/download/. Having installed the Omnis App Server you will need to serialize it according to the number and type of clients you expect to serve. There are a number of different server deployment licenses for running web and mobile apps in the JavaScript Client, or you may have a server license included in the Community Edition. Please contact your local sales office for details about these Omnis App Server deployment licenses. In addition, you will need to purchase a different development license and deployment server license to create and deploy standalone mobile apps running in serverless client mode.

Licensing Mechanism

In order to enforce licensing for JavaScript Client based apps, the UUID of each client is logged with the Omnis App Server. Prior to Studio 8.1.6, the UUID was stored in a cookie in the client computer which required any clients to have cookies to be enabled for this licensing mechanism to work. However, the method for storing the client UUID has changed in version 8.1.6: the UUIDs are now stored in the 'localStorage' on each client which is now used to manage client licenses on the Omnis App Server. Therefore, clients no longer have to have cookies enabled for App Server Licensing to be enforced.

Omnis Web Architecture

The server side of your web or mobile app comprises the Omnis App Server that runs your Omnis library, a standard Web Server, and your database server(s). All these parts would either run on the same machine, or more typically would be on the same LAN or subnet, communicating via TCP/IP. The web server and the Omnis App Server can be hosted on a Windows, macOS, or Linux computer server.

The web server would store your entire website, including any HTML pages containing your JavaScript remote forms and any other web pages as required, such as landing pages.

The Omnis library contains the GUI and data class definitions, business rules, and application logic, while the database server would be an industry-standard database server, such as Oracle, MySQL, PostgreSQL, Sybase, DB2, or any JDBC or ODBC compliant database such as MS SQL Server.

The Omnis App Server is the main engine that runs your web and mobile applications. It is a multi-threaded server that runs your Omnis application, executing all the business logic, accessing your server database(s), and handling all the client interactions to-and-from your web and mobile clients. Web and mobile client access to the Omnis App Server is restricted to a specified number of users and is determined by the server license, which you must purchase separately from your locale sales office.

Editing Your HTML Pages

Omnis creates an HTML page when you test your JavaScript remote form (when you press **Ctrl-T** or use the **Test Form** option) that contains details of the JavaScript Client object, your remote form class, the location of your web server (or your computer when testing), and so on. You can use this HTML file for deploying your web application, but you will need to edit it, or copy the relevant lines of code containing the JavaScript Client object to your own HTML pages.

The test HTML file has the same name as your remote form plus the .htm extension, and is located in the **html** folder under the main Omnis folder. For example, under Windows the HTML template is located in your AppData\Local folder, such as:

```
C:\Users\<user-name>\AppData\Local\Omnis Software\OS<version>\html
```

You can edit the test HTML in a standard web page design tool or a text editor, such as Notepad under Windows.

The JavaScript Client Object

In the template HTML file, there is a <div> that contains the JavaScript Client called 'omnisobject1'. It contains various parameters that provide details about your application that are sent to the Omnis App Server when the client connects. The parameter names are in lowercase and prefixed with 'data-', to comply with HTML5.

The code for omnisobject1 from the 'jsctempl.htm' file is as follows (for existing users, the equivalent old parameter names are included in parenthesis):

```
<div id="omnisobject1" style="position:absolute;top:0px;left:0px" data-webserverurl="" (was WebServerURL)
data-omnisserverandport="" (was OmnisServerAndPort)
data-omnislibrary="" (was OmnisLibrary)
data-omnisclass="" (was OmnisClass)
data-param1="" data-param2="" (was param1, param2,..)
data-commstimeout="0"> (new parameter)
```

For example, the HTML page containing the Omnis quiz has the following <div> tag containing the JavaScript Client object and its parameters:

```
<div id="omnisobject1" style="position:absolute;top:0px;left:0px" data-webserverurl="http://194.131.70.208/cgi
```

You can add another Omnis object to the same HTML page, but it must have a unique id, such as "omnisobject2", and you can set its own server, library, and form parameters.

JavaScript Client Object Parameters

data-webserverurl

The *data-webserverurl* property identifies the HTTP URL of either the local IP address of your computer during development or the address of the Web Server and location of the web server plug-in.

When testing with the development version of Omnis, the data-webserverurl is set to "_PS_" which will be replaced with the address of the computer from which the HTML page is being served, that is, your development computer. For example, if the test HTML page is at http://127.0.0.1:5000/jschtml/test.htm, then _PS_ will be replaced with http://127.0.0.1:5000. For testing, data-omnisserverandport will be empty.

For deployment, when using a Web Server, the data-webserverurl parameter should be set to the location of the Omnis web server plug-in, such as http://www.myhost.com/scripts/omnisapi.dll, .which handles all the communication between the server and the client.

data-omnisserverandport

For development and testing *data-omnisserverandport* can be empty. For deployment, data-omnisserverandport tells the Omnis web server plug-in how to connect to the Omnis App Server. If the Omnis App Server is on the same machine as the web server, then data-omnisserverandport can be the port number of the Omnis App Server, e.g. it could be "5000" if the Omnis App Server is at port 5000 on the same machine as the web server. If the Omnis App Server is on a different server from the web server, then the data-omnisserverandport parameter must be "IP-Address:Port-number" of the Omnis App Server, e.g. it could be "111.222.000.111:5000" if the Omnis App Server is at port 5000 on a machine with IP address 111.222.000.111.

data-omnislibrary and data-omnisclass

The *data-omnislibrary* parameter identifies the library containing your remote form, and *data-omnisclass* is the remote form class displayed by the omnisform object.

data-commstimeout

The *data-commstimeout* parameter allows you to give the end user the option to timeout a request or carry on waiting. The default value of zero means that no timeout is applied, and the client will continue to wait for a response. To apply a timeout, you need to enter an integer representing the timeout in seconds. When the client sends a message to the server it must respond within this timeout period, otherwise the user will be prompted to either continue waiting for a response, or abort the request.

Version and Build Number

The **Version** and **Build** number of Omnis Studio is included in the HTML page. The "%%version%%" placeholder will be replaced with the Omnis Studio version number when the Test Form option is used. For example, the following is added to the beginning of the html:

```
<!DOCTYPE html>
<!-- Generated by Omnis Studio Version 11.0 Build 110034477 -->
```

The "%%build%%" placeholder will be replaced with the Omnis Studio build number, e.g. 110034477 in the above example.

Managing Wifi Connections

The client should handle a wifi connection going away while processing a request at the server, so when the connection comes back, the client should respond properly. Otherwise, you have the option to provide a timeout, as above.

Additional and Custom Parameters

You can specify up to nine additional parameters (named data-param1, data-param2, ..) which can be passed into the row variable parameter pParams of the $construct() method of the remote task assigned to the remote form.

In addition to the pre-defined parameters, you can include your own custom parameters if you wish to pass extra values to the task or form $construct method. Custom parameter names should be prefixed with "data-" and should be lowercase. The parameter will be added to the construct row variable which you can interrogate in your task or form $construct method.

Positioning the Omnis object

The position or alignment of the “omnisobject1” itself within your HTML page is where the Omnis JavaScript Client remote form will be displayed in a web browser. For mobile clients you have less control since the remote form will usually fill the entire mobile screen. By default, the JavaScript Client is displayed in the top left corner of the client browser (set using style=“position:absolute;top:0px;left:0px”), but for web clients you can reposition the JavaScript Client object to appear anywhere in your HTML page using alternative style or positioning parameters.

Centering the omnisobject

By replacing the default style parameter in the div tag on the JavaScript Client object with style=”width:900px; margin:auto” will center the remote form in the browser, assuming your remote form is 900 pixels wide. Note that you cannot use the align property on the omnisobject to reposition it, therefore align=“center” will not center the omnisobject.

The above technique for centering the omnisobject will be fine for web-based applications but it may not work for all mobile devices, therefore you can use the following method to center the omnisobject for all devices. You have to place two extra divs around the div containing the omnisobject and change the style parameter inside the omnisobject div, as follows:

```
<div id="grandparent" style="float:left; width:100%; overflow:hidden; position:relative;">
  <div id="parent" style="clear:left; float:left; position:relative; left:50%;">
    <div id="omnisobject1" style="display:block; float:left; position:relative; right:50%;" data-webserverurl=
    </div>
  </div>
</div>
```

CSS styles and JavaScript folders

The Omnis **html** folder contains a number of other folders, including **CSS**, **formscripts, icons, images,** and **scripts**, which contain the CSS style sheets, JavaScript files, icons, and other images, which are required to run the Omnis JavaScript Client: *these folders and their contents* ***must be copied*** to the equivalent location in your web server relative to the HTML page containing your remote form(s).

Icon Sets

You need to copy any icon sets used by your JavaScript Client application, including any of your own icon sets as well as the Omnis supplied icon sets, from your development tree to two places:

- The ‘html/icons’ folder of the **Omnis App Server** (so Omnis can generate relative paths for the icon URLs).
- The ‘icons’ folder at the same location as the .htm file containing the remote form for your app on your **Web Server** (these are the actual resource files that the client will request).

Any custom icon sets you have used should be named in the $iconsets property of your library. The location of these icon sets in your development tree could be in one of the following locations

```
<Application Directory>/iconsets
<User Data Directory>/iconsets
<User Data Directory>/html/icons
```

For example, if you have used the ‘studio’ icon set, you might copy this from:

```
<Dev Application Directory>/iconsets/studio
```

to:

```
<App Server's User Data Directory>/html/icons/studio
<Web Server Location of .htm file>/icons/studio
```

Fav icon

The icon used for the test HTML page (displayed in the top-left of the browser tab) is the image file called 'favicon.ico' located in the html/images folder. The image file in the development version of Omnis is the Omnis logo, but for deployment you can replace this image with your own fav icon file.

Customizing the JavaScript Working Message

You can change the appearance and positioning of the working message displayed in the JavaScript Client when some processing is occurring. The working message is a transparent overlay with a circular spinner which is laid over the main JavaScript Client area. You can restyle the working message overlay by extending the 'standardOmnisLoadingOverlay' class in the user.css file found in the html\css folder in your Omnis development tree. This follows the same structure as the CSS class parameter for the "showloadingoverlay" client command: see Custom Loading Indicator.

Setting up the Omnis App Server

In order to deploy your Omnis web or mobile application, you need to download and install the **Omnis App Server** which will run your Omnis application (library) file. The Omnis App Server is available for Windows (32-bit & 64-bit), macOS, and Linux servers. There is a Tech note on the Omnis website about setting up the Omnis Server which contains all the latest settings, etc:

- **TNJS0003:** Setting Up The Omnis App Server

On Windows, you can set up the Omnis App Server to run as a Service which is described in this Tech note:

- **TNWI0002:** Running the Omnis Application Server as a Windows Service

You should read these tech notes for the latest information about setting up the Omnis App Server, including information about the latest version of IIS and Apache appropriate to deploying Omnis web and mobile apps.

The server does not need to be located on the same machine as your web server. For testing and debugging you can use the Omnis development version, but for deployment you must use the Omnis App Server.

You need to place any libraries containing your application in the 'Startup' folder within the Omnis App Server tree, so that Omnis automatically opens them when it starts up.

When Omnis is running as a Service, the prompt for a serial number is not shown on startup, plus serialization errors are sent to the Windows Application event log.

Server Configuration

There is a JSON based configuration file in the **Studio** folder called **'config.json'** which is used to configure the Omnis App Server, including setting up properties for the server itself and logging, as well as the settings for Web Services support. The config file also includes a section to enable the Java Class cache to be cleared, and other configurable items in Omnis.

The configuration of the Omnis App Server can be set up during installation or by selecting the **Server Configuration** option in the File menu in the Omnis App Server. Alternatively, you can configure or change the settings of the Omnis App Server by editing config.json using any compatible text editor, but the file must conform to JSON syntax.

Server Configuration File

The first part of the config.json file for the Server has the following layout:

```
{
  "server": {
    "disableInRuntime": false,
    "port": "",
    "stacks": 5,
    "timeslice": 20,
    "webServiceURL": "",
    "webServiceConnection": "",
    "webServiceLogging": "off",
    "webServiceLogMaxRecords": 100,
    "webServiceStrictWSDL": true,
    "headlessAcceptConsoleCommands": false,
    "headlessDatabaseLocation": "",
    "service": "homnis",
    "start": false,
    "retryBind": true,
    "showBindRetryMessage": true,
    "bindAttempts": 0,
    "runtimeOpensTraceLogOnSocketBindError": true,
    "RESTfulURL": "",
    "RESTfulConnection": "",
    "autoChunkRESTfulURLs": [
      "http://localhost:8080/omnisrestservlet"
    ],
    "getpdfFolders": [
      ""
    ],
    "overridePushURL": "",
    "timeOffsetMinutes": 0,
    "timeoutReads": true,
    "readTimeout": 20
  },
```

where

- **port, stacks, timeslice**
  configure the Omnis App Server executable
- **disableInRuntime**
  when set to true (default is false) prevents the Omnis Server listening on its own port: this can be used to prevent firewall prompts when the Omnis Server is not required
- **webService**...
  these parameters configure WSDL/SOAP based web services
- **RESTful**...
  these parameters configure REST based web services
- **start**
  if true means Omnis automatically executes Start server at startup
- **retryBind**
  Set retryBind to false if you do not want Omnis to retry binding to the server port after its first attempt; retryBind defaults to true if it is omitted
- **showBindRetryMessage**
  If retryBind is true (or omitted), showBindRetryMessage controls whether or not a working message is displayed while retrying the bind to the server port
- **bindAttempts**
  If retryBind is true (or omitted), a positive value of bindAttempts overrides the default number of attempts to bind to the port at 1 second intervals
- **timeOffsetMinutes**
  allows you to add an offset to the date-time setting on the Omnis App Server. Omnis adds the value of this setting to the current system date-time when generating the value for #D and #T. If the entry is not present, it defaults to zero, meaning no offset is applied

Server Logging

The Omnis App Server has a logging mechanism to support RESTful web services, if applicable. There is an external component that performs logging, located in the logcomp folder of the Studio tree, with just one component, logToFile. NOTE: you can use logToFile in the Development version of Omnis Studio to log server activity including REST calls which can be useful while testing & debugging your app.

You can configure server logging by adding a member to the config.json configuration file, with the following layout:

```
{
  "server": {
    "//": "See Server Configuration section above",
  },
  "log": {
    "datatolog": [
      "restrequestheaders",
      "restrequestcontent",
      "restresponseheaders",
      "restresponsecontent",
      "tracelog",
      "seqnlog",
      "soapfault",
      "soaprequesturi",
      "soaprequest",
      "soapresponse",
      "cors",
      "headlessmessage",
      "headlesserror",
      "systemevent"
    ],
    "logcomp": "logToFile",
    "logToFile": {
      "stdout": false,
      "folder": "logs",
      "rollingcount": 10,
      "daily": true
    },
    "overrideWebServicesLog": true,
    "windowssystemdragdrop": false
  },
```

where

- **logcomp**
  is the name of the logging component to use, that is, “logToFile” which referencese the logtofile.dll component in the logcomp folder of the Studio tree.
- **datatolog**
  is an array that identifies the data to be written to the log - one or more ofthe values listed in the array above
- tracelog means that data written to the trace log is also written to the new log
- seqnlog means sequence log entries that record method execution are written to the new log instead of the old sequence log file
- **overrideWebServicesLog**
  allows you to just send SOAP web service log entries to the new log; true means just send log entries to the new log, false means send them to both the old web services log and the new log.

- **windowssystemdragdrop**
  enables system drag and drop behavior available in versions before Studio 10.2
- **logToFile**
  is a member with the same name as the value of logcomp. This contains configuration specific to the logging component.
- **folder**
  is the name of the folder where logs will be placed; this can be a single folder name relative to the Omnis data folder, or it can be a full path name, which must not end in a path separator character
- **rollingcount**
  is the number of log files that will be maintained, can be up to 1024. The log component uses a new log file every hour (and a new one at startup). The log component deletes the oldest file or files so that the number of log files does not exceed this count
- **daily**
  allows you to enable daily log file (true) or the defauly hourly log files (false).
- **stdout**
  for the MPS, if set to true, logging from all processes in the MPS (main and child) will go directly to standard output, serialised between all of the processes using a shared mutex

logToFile: Folder location

The "folder" item in the "logToFile" section can be a folder name (relative to Omnis folder), or a full path name, which must not end in a path separator character, and the end folder name will be created if it does not already exist. In previous versions, you could only specify a folder relative to the Omnis folder, but now a full path can be used which can be outside the main Omnis folder. For example:

```
  "folder": "/Users/bd/Sites/logs"
```

would send the log to the specified folder, while

```
  "folder": "logs"
```

would still be read as relative to the main Omnis folder (note no starting or ending path separator).

You must use / as the path separator on macOS and Linux, whereas, you can use / or \ on Windows.

logToFile: Rolling count

The maximum for the "rollingcount" item in the "logToFile" can be an integer up to 1024. The logtofile component uses a new log file every hour, so the new max value would allow logs to be stored for up to six weeks, at which point the oldest logs would be deleted.

If there is an error initialising logging, the logtofile component also writes it to standard output when running on Linux.

logToFile: daily

If set to true, Omnis creates a new log file for each day. Omnis re-uses the log file for a day if it is already present at startup. The rollingcount applies as for hourly logs. The item defaults to false, which means hourly logging is used.

Log File Format

Each log record has the following layout:

```
{"thread":0,"when":"20141017 14:04:14","type":"tracelog","length":127}ExternalLibrary File 'C:\dev\UnicodeRun\
```

where

- **thread**
  identifies the thread logging the entry,

- **when**
  is the date and time of the entry,
- **type**
  is the type of the entry (one of the datatolog values), and
- **length**
  is the length in bytes of the data following the initial JSON header.

This is followed by a final CRLF. Log files can typically be read in a text editor, but be aware that they can contain binary data if the content of RESTful requests or responses is binary.

Setting the Omnis App Server Port Number

You can set the Omnis App Server port using the command *Calculate $prefs.$serverport as 5912*.

Server Multi-threading

The Omnis App Server allows multiple client requests to be processed concurrently, allowing smoother allocation of available processor time and avoiding any lengthy delays on the client: a *client request* might be a request from a client to execute an event, or a request from the client to call a server-side method. To handle these multiple client requests, the Omnis App Server can be made to run in multi-threaded mode. By default, the Omnis App Server runs in single-threaded mode, handling client requests in a strictly first-come, first-serve basis; in this case, client requests are queued, with each request being handled only when the previous request has completed. You can however handle multiple client requests concurrently using the Omnis Multi-threaded Server, which you can enable by executing the *Start server* command on the Omnis App Server: see below.

The Multi-threaded Server maintains a pool of method stacks that can process web and mobile client requests simultaneously. The pooling mechanism allows a balance to be struck between performance and server resources - the number of method stacks in the pool is configurable via the $serverstacks Omnis preference, and also available in the Omnis App Server Configuration dialog.

Multiple Method Stacks

The standard single-threaded Omnis App Server has only a single method stack to process methods. Broadly speaking, once a method call has been pushed onto the method stack no other method call can begin to execute until the first call has completed. For the majority of web and mobile client applications this is fine for processing events, particularly if some processing is performed on the client and your web server receives relatively few hits or requests for data. By contrast, the Omnis Multi-threaded Server contains a pool of method stacks which are available to process multiple client requests, and this is appropriate for more data intensive web applications where lengthy calls to a server database are required, or for web applications that receive higher volumes of traffic. When a request to execute a method is received from a web or mobile client, that method call is pushed onto any unused stack or, if there are no unused stacks, the message is queued until one becomes available. Each method stack runs in its own thread, which means that if a method stack is stalled (for example, it is waiting for the database server) the other stacks will continue to execute.

Potentially, the Multi-threaded Server may have to cope with a very large number of simultaneous clients, each with their own remote form and remote task instances. Typically though, a small proportion of clients will require the use of the server at any one time. In fact, multi-threading does not increase the server processor time available, it just allows the available processor time to be allocated in a smoother way. The method stack pool mechanism allows a balance to be struck between performance and server resources - the number of method stacks in the pool is configurable with the $root.$prefs.$serverstacks property, which is set to 5 by default.

As method stacks are allocated dynamically, it is very likely that a remote client will not get the same method stack every time it executes a method on the server. Each method stack contains its own state which, apart from during an individual method call, does not belong to any particular client. This state includes the Current Record Buffers (CRBs) for all files and variables (apart from class variables) and such modes as the current list. A client cannot rely on any properties or values of this state being preserved across different method calls. The only things belonging to the client are its instance and task variables. So a client must do such things as setting the main file and current list each time one of its methods is executed, and should not rely on such things as the values of memory-only fields being maintained across method calls. As a special case, the class variables for the remote task and remote form classes are shared amongst all clients so can be used to hold shared data (see below for the warnings about the care needed when using shared variables).

Using the Multi-threaded Server

When the Multi-threaded Server starts up, it opens the libraries, data files and SQL session pools required by the clients (see below for the description of a SQL session pool). You need to issue the *Start server* command to cause the method stacks and associated threads to be created. The *Start server* command can specify an optional stack initialization method; when specified, this method is pushed onto every client method stack and allowed to execute (so if $serverstacks is 5 it will execute five times), so it can be used to initialize the state of the method stacks. The *Start server* command generates a fatal error if, due to lack of resources or some other reason, it is unable to complete successfully.

When you want to stop the server, you should issue the *Stop server* command, but quitting the Studio program achieves the same result.

When the server is active, Omnis continues to be responsive to events on the server and could, for example, display a window with 'Start server' and 'Stop server' buttons. It is not recommended that the server program performs any substantial tasks while it is deployed and in use listening for client requests.

Any runtime errors generated by client methods are reported in the trace log (using a similar mechanism as errors during library conversion), but you can override this default behavior by making sure each client method stack has an error handler. You can use the stack initialization method call for the *Start server* command to define an error handler for each method stack.

Note that you cannot debug methods running in a remote form or task instance, after you have called the *Start server* command in a development version of Omnis.

Database Access

If you are accessing a server database in your web application, and using the Multi-threaded Server, you must use the Object DAMs (introduced in Omnis Studio 3.0) which are capable of multi-threading. Using the Object DAMs, you can connect directly to Oracle, MySQL, PostgreSQL, DB2, and Sybase, as well as most ODBC- and JDBC-compliant databases such as MS SQL Server.

The Object DAMs are implemented as external components and use object variables based on the Object DAMs, and interact with a DAM using the methods of the object. Using this approach, you create object variables of a particular DAM class. There is a group of common methods that apply to all DAM objects and a set of DAM specific methods based on the type of object. Various chapters in the *Omnis Programming* manual provide more information about accessing your data using the DAMs.

Omnis App Server Commands

There are some Omnis commands that you can use to control the Multi-threaded Server.

Start server

The *Start server* command is used to create the client method stacks and associated threads. It takes an optional stack initialization method as a parameter. The command clears the flag if it is used in a copy of Omnis which is not capable of supporting multi-threading or your serial number does not allow clients to connect. A fatal error is generated if for some other reason it is not possible to create the stacks and threads.

Stop server

The *Stop server* command stops the server from responding to client requests. Once the server has been started you should stop it before quitting Omnis, before using Omnis for anything apart from serving client requests (e.g. running a standard LAN-based Omnis application), or before opening or closing any Omnis data files or libraries.

The *Stop server* command disposes of all remote task and form instances. The resources used by the client stacks and threads are not released, but they will be reused by the next *Start server* command.

Begin and End critical block

These commands are used to denote a section of code which needs to execute in single threaded mode without allowing other client methods to execute. For example:

```
Set current list cList
Begin critical block
  Build list from file
End critical block
```

Here cList is a class variable which is shared amongst the clients and the critical block is used to prevent other clients from accessing the list whilst it is being built. Generally class variables should only be used when the shared functionality is essential and only with care:

```
Calculate cString as 'abc' ;; OK
Calculate cString as $cinst.$xyz()
# only OK inside a critical block
```

Simple atomic operations such as the first line of the above example are safe, but when a method call is involved it may be interrupted by other threads and cause problems. Class variables should not be used as bind variables or as the return list for SQL operations.

Yield to other threads

The *Yield to other threads* command is a hint that the executing thread is waiting for other threads and is prepared to yield its processor time. It can be used when waiting for semaphores (since with the Multi-threaded Server another client stack could be holding the semaphore), as follows:

```
Do not wait for semaphores
Repeat
  Prepare for edit
  If flag true
    Break to end of loop
  End If
  Yield to other threads
Until break
```

Commands which are not available to a client

The following commands are not available for methods running on the Multi-threaded Server. They usually generate a ‘Command not available when executing a client method’ fatal error but some (such as the Debugger… group) simply do nothing:

- The Libraries… commands
- The Classes… commands
- Pre V30 SQL commands…
- The Data management… commands
- The Message boxes… commands
- The Debugger… commands
- Quit Omnis
- Enter data
- Prompted find
- And all the Omnis data file and lookup commands

Any other command which would cause a dialog to be displayed on the server is not available for methods running on the Multi-threaded Server. In addition, there is a lot of notation, such as the notation for opening and closing libraries and data files, that will not work in a method running on the Multi-threaded Server.

SQL Session Pools

Suppose the Multi-threaded Server has N method stacks, and therefore N threads capable of processing methods on behalf of clients. This means that at any one time, there can be at most N SQL sessions in use. However, the Multi-threaded server may have many more than N current users. If you are using SQL, you need a potentially large number of sessions to the database server. There is nothing wrong with this in itself, and there are occasions when you might want to use database access permissions, to control the tables and columns accessible to different users. If this is the case, you require a separate session for each user. However, if all users have the same access permissions, you really only need N SQL sessions. This can significantly reduce the resource usage of the server. SQL Session Pools provide a way to do this.

A SQL Session Pool is a set of multi-threaded DAM sessions, which can be shared by clients. Typically, you would create a session pool with one session for each method stack. Details of how to use SQL Session Pools can be found in the *Omnis Programming* manual.

Server Load Sharing

Load sharing allows a pool of Omnis App Server processes, running on one or more machines, to serve clients. Once a client connects to an Omnis App Server process, all subsequent requests for that client need to be handled by the same Omnis App Server process, since the process contains the instance data for the client. Therefore, load sharing provides a mechanism that assigns a new client connection to an Omnis App Server process. *Note that Load Sharing is not available in the Community Edition.*

The data-omnisserverandport parameter in an HTML page normally has the syntax:

```
[(IP address|domain name):](service name|port number)
```

To use load sharing, you prefix this property with a name for the pool of Omnis App Server processes and a comma, for example “Omnis,6000”, or “Omnis,194.131.70.197:6000”. In this case, the address information in the property no longer addresses an Omnis App Server. Instead, it addresses a new module, a load sharing process.

When a new connection arrives at the Omnis web server plug-in, the plug-in inspects the syntax of the data-omnisserverandport parameter. If it is prefixed by a pool name, the plug-in connects to the load sharing process, and sends it a message that asks for the address of a server process in the pool. The load sharing process typically returns the address and port number of the least busy process in the pool. The plug-in then connects to this process, and sends the web or mobile client connection to it. When the plug-in responds to the client, it includes the address of the Omnis App Server process in the response.

When the client sends subsequent messages to the web server for this web or mobile client connection, it sends the address passed in the connect response instead of the data-omnisserverandport parameter. Thus the only additional overhead imposed by load sharing occurs during connection setup.

So how does Omnis know (1.) which Omnis App Server processes exist, and (2.) which Omnis App Server process is the least busy? The load sharing process (LSP) has a .ini file, which contains the pool names for the pools for which it is responsible, and for each pool, the addresses of the Omnis App Server processes in the pool. Periodically, the load sharing process polls each Omnis App Server process, and asks it for the percentage of web or mobile client connections currently in use (using the serial number as the maximum), and information about how much time the server has spent processing requests. The load sharing process combines this information to determine which process is the least busy.

You can configure the time interval between polls of each Omnis App Server process via the .ini file. Once every 10 or 20 seconds is usually frequent enough.

Enabling Load Sharing

To enable the load sharing process you need place the LSP program on your web server, or a machine connected to your web server. It is a single executable called Omnislsp.exe (Windows) or omnislsp (Linux and macOS). A configuration file (omnislsp.ini) must be made to accompany the Omnislsp program, and this takes the following format:

```
[Setup]
Port=6001
QuietMode=0
BucketSize=100
LogLineThreshold=16
Pool1=Omnis
[Omnis]
PollTimer=10
Server1=123.145.71.123:7001
Server2=123.145.71.124:7002
```

The commands for the lsp are:

```
omnislsp -start

omnislsp -stop
```

(with the omnislsp.ini in the same directory as the program)

The Port entry in the .ini file identifies the TCP/IP port number on which the LSP listens for requests from the Omnis web server plug-in.

The QuietMode entry in the .ini file indicates if the LSP generates OK messages, or messages to the console, to report its status. When set to zero, it will generate messages. When set to one, it will not.

The LogLineThreshold entry in the .ini file indicates when the text log generated by the LSP will be reduced in size. If the LSP writes a line to the log, and the file contains LogLineThreshold lines, it will reduce the file size to LogLineThreshold/2 lines, maintaining the most recently written lines, before writing the line to the log. The log is in the same directory as the omnislsp program.

The BucketSize entry specifies how the LSP breaks up the server processes into groups, based on how busy they are. It is a value in milliseconds. The LSP divides the processes into 10 buckets, based on the average time to process a request obtained from the information it gathers by polling the server processes periodically. The buckets are numbered 1-10, where 1 contains the least busy servers, and 10 the most busy servers. A server is in the smallest numbered bucket, for which its average time to process a request is less than or equal to (bucket number)*BucketSize. If a server is so busy that this calculation does not allocate it to a bucket, it belongs to bucket 10. You may need to experiment with possible settings for BucketSize, in order to determine the optimum setting for your application.

Each pool has its own section in the .ini file. The PollTimer entry indicates the frequency in seconds at which the LSP polls the server processes in the pool for information. The ServerN entries identify the TCP/IP address and port of each server process in the pool.

You also need to edit the 'data-omnisserverandport' parameter in your HTML file containing the JavaScript Client plug-in, for example:

```
data-omnisserverandport="Omnis,6001" or
data-omnisserverandport="Omnis,123.456.789.010:6001"
```

where Omnis is the name of a pool of Omnis App Server processes and 6001 is the port number of the LSP.

On the LSP servers

The Omnis App Servers may be stopped and restarted without the need to stop the LSP.

Load Sharing Mechanism

The load sharing process periodically polls the processes in a pool of Omnis App Server processes. Each server returns the current number of connections to the server, the maximum number of concurrent connections allowed to the server (specified by the serial number), the number of requests since the last poll, and the total elapsed time in milliseconds taken to process the requests. The load sharing process organizes the servers into buckets, based on the results of the information returned from polling the servers.

When a connection request arrives at the load sharing process, it allocates a server to the request as follows. It traverses the buckets, starting with that for the least busy servers, looking for a server that has some free connections. Within a bucket, it looks for the server with the smallest percentage of connections in use, using the results of the last poll. If there is more than one server with the same smallest percentage of connections in use, the process allocates the connection to the server to which it least recently allocated a connection. At this point, the load sharing process also updates the connection statistics from the last poll, to reflect the new connection. The traversal stops when a free process has been found. If all servers are fully utilized, the LSP allocates the connection to a server at random; in this case, it is likely that the server will reject the request, and return a suitable error to the client.

Installing as a Service (Windows only)

The omnislsp process can be installed as a service which starts-up automatically when Windows loads. For this purpose, two additional parameters are supported:

omnislsp –install Creates and starts the “Omnis Load Sharing Process” service.
omnislsp –uninstall Stops and removes the service.

The startup-type for the new service is set to “Automatic” and the service uses the omnislsp executable and .ini file at their current locations. When omnislsp runs as a service, dialog boxes are disabled and messages are written to the application event log instead.

LSP Debugging

You can enable debugging in the LSP using the DebugMode setting in the [Setup] section of ini configuration file. If DebugMode=1, more information will be logged, such as when the LSP fails to connect to the Omnis App Server, in the following format:

Tue Sep 13 20:38:19 2023 [DEBUG] [127.0.0.1:7001] Failed connecting to OMNIS server.

Furthermore, any debug messages will have [DEBUG] in the message.

Socket Binding

For the development version, Omnis will retry the bind 5 times, once a second, and then report an error via the trace log; this behavior is the same as previous versions. The Omnis App Server will retry indefinitely once a second.

When running as a Service, after a few bind attempts, Omnis outputs a message to the system event log (if the event log is being used for Studio), "Failed to bind web client socket - will retry indefinitely". If the bind is eventually successful, Omnis outputs a second message, "Successful bind of web client socket".

Note that you can stop the service while the indefinite retries are occurring, but you cannot use the tray to bring Omnis to the front (or do anything else with the Studio service) as the web client bind occurs quite early in the initialization of Studio.

The Omnis App Server displays a working message while the retries are occurring; this allows the end-user to cancel the retry loop.

Managing Server Timeouts

You can manage what is displayed in the end user's browser when the Omnis Server responds with a Server error or Disconnected message. You can create a client-executed remote form method named $ondisconnected which will be called when there is an error on the server or the client is disconnected.

The method has a single parameter which provides the error text. This is only populated if it was triggered by a server error, rather than a disconnect due to Remote Task timeout etc. If you wish to prevent the default behavior, you must return kTrue from this method.

The form which initiated the server request will be queried for the method first. If it is not found, or it does not return kTrue, any parent forms (if it is a subform) will be tried.

Setting Up Your Web Server

In addition to the Omnis App Server, you need to install and configure a standard web server, such as Microsoft IIS, Apache, or Tomcat: you should download the web server software from the appropriate vendor. You need to install the appropriate Omnis ***Web Server plug-in*** in the correct location on your web server, for example, in the /cgi-bin, /scripts, or /webapps folder. A web server plug-in is available for Apache on Windows, macOS and Linux, plus IIS and Tomcat under Windows.

Installing the Web Server Plug-in

Typically, a web server will have a place to store all executable code accessible via HTTP over the Internet. This is often the /cgi-bin or /scripts folder, but it can be any folder configured to allow execution. You need to place the Omnis Web Server plug-in, such as the omnisapi.dll (IIS) or mod_omnis.so (Apache), in this folder and ensure that the web server is set up correctly to enable it to be executed. The location of the plug-in should be specified in the data-webserverurl parameter of your HTML files containing your remote forms.

For information about installing the Omnis Web Server Plug-ins, please refer to the Tech note: **TNJS0003** Setting Up The Omnis App Server

Installing the Java Servlet

NOTE: In previous versions of Omnis Studio (prior to Studio 10.x) we included a JavaServlet plug-in, but this is no longer provided in the Omnis tree due to a change in licensing for Java: see the Readme for more details about Java Legacy Integration or contact Omnis Tech Support for more information. The information here is provided if you need to use the JavaServlet, but otherwise should be ignored.

The Java Servlet web server plug-in allows you to run your Omnis web applications with any web server that supports version 2.3 of the Servlet API. Web servers that comply with Java Servlet API 2.3 include the Apache TomCat and Jetty web servers, but there are several others. For more information about Java Servlets, see: https://www.oracle.com/java/technologies/

The Java Servlet allows Omnis remote forms to connect to an Omnis App Server via a Java web server. In this respect, the Servlet is the Java equivalent of the ISAPI, CGI or the regular Apache plug-in available in Omnis Studio.

The Java Servlet is called RdtaServlet and can be found in the Omnis Studio installed tree in the clientserver\server\omnisservlet folder. Inside the Servlet folder, the servlet files are in the following folder structure:

```
Servlet
|
 - WEB-INF
   |
    - rdtaserv.dll (Platform dependant)
      |
       - web.xml
         |
          - classes
            |
             - com
               |
                - rdta
                  |
                   - RdtaServ.class
```

To install the servlet on your web server, such as the Apache Tomcat server, you should place the whole servlet folder in the Webapps folder within the Tomcat installation. If you wish, you can rename the servlet folder.

You will need to restart the webserver in order for the servlet to be loaded.

Web server parameters for the Java Servlet

The data-webserverurl parameter in your HTML containing your remote form must have the correct format to access a web server with the Java Servlet.

- **data-webserverurl**
  this must be the URL to the tomcat webserver appended with the name and path of the servlet, such as "http://www.mydomain.com/servle

Note if you have renamed the servlet folder you should use the new name in the data-webserverurl parameter instead of 'servlet';

The default value for %servletname% is 'rdtaservlet'. You can modify this or add others by modifying the web.xml file. The following xml can be added to the <web-app> element in the document to add a new servlet alias name. For example

```
  <servlet-mapping>
    <servlet-name>RdtaServ</servlet-name>
    <url-pattern>/%servletname%</url-pattern>
  </servlet-mapping>
```

ISP Web Hosting

If your web site is hosted by a third-party, such as an ISP, they will need to place the Omnis **Web Server Plug-in** in their cgi-bin folder, and furthermore they need to provide you with a direct connection to the Internet. Your ISP may want to test the web server plug-in, usually the case for any files you place in their cgi-bin folder. Alternatively, you can rent a whole server in the cloud, in which case you will have more control over the setup of the server and you can place the Omnis files in the correct location.

If your Omnis web application uses a web site hosted by an ISP you will need to adjust your port settings in the Omnis App Server and HTML files. In this case you can use DomainName:Port or IPAddress:Port in your port setting.

Secure Sockets (SSL)

It is regarded as essential to use an SSL certificate for all web and mobile applications, to protect communications between clients and the application server, and indeed most browsers will mark any web pages (applications) that do not use SSL as unsafe.

You can use secure sockets (HTTPS) if you have installed an SSL certificate on your web server. The JavaScript Client will use a secure connection to connect the client to the web server if you prefix the URL or IP_address in the data-webserverurl parameter with "https://", for example:

```
https://remainderOfFullURL
```

In addition, remote tasks have the $issecure property that lets you turn secure mode on and off dynamically, by assigning to the property at runtime in your application.

Secure socket connections do not support the "address:port" format for the data-webserverurl parameter of a JavaScript Client object.

Web Server Plug-in Custom Configuration

You can configure the Web Server plug-in via a separate configuration file, allowing greater security and control over user access to your Omnis web or mobile application. It is not necessary to configure the web server plug-in in most circumstances, but this functionality provides added flexibility and security, as follows:

- **Access and Security**
  Using the web server plug-in configuration file you can restrict access to an Omnis App Server
- **Configure Server parameters**
  The configuration file allows you to override parameters for the Omnis App Server, or in effect, provide default parameters if the oserver parameter is blank in your cgi parameters in your html; in this case the value would be taken from the configuration file.
- **Post content to Remote tasks**
  the configuration file allows you to pass HTTP Post content to the Omnis remote task.

The parameters specified in your configuration file can provide default connections for clients, simplifying the post command required to connect to the Omnis App Server. ***The configuration file should be placed in the same directory as your Web Server plug-in.***

Server plug-in activation

The custom functionality in the Omnis Web Server plug-in is built into the plug-ins supplied with the Omnis App Server installation, but ***to activate the functionality you have to rename plug-in itself.***

mod_omnis.so

If you are using mod_omnis.so under Linux or Windows, you need to change the value of the location in your http.conf or equivalent apache configuration script to /omnis_apacheini, for example:

```
<location /omnis_apacheini>
  SetHandler omnis-apache
</location>
```

nph-omniscgi

Rename the nph-omniscgi.exe to nph-omniscgiini.exe for Windows, or rename nph-omniscgi to nph-omniscgiini for Linux.

omnisapi.dll

For Windows IIS based servers, rename omnisapi.dll to omnisapiini.dll.

rdtaserv.dll

If you are using the Web Services enabled Web Server plug-in, rename rdtaserver.dll to rdtaserverini.dll.

Creating a Configuration file

The configuration file should be named omnissrv.ini and be placed *in the same directory as your Web Server plug-in,* for both Windows and Linux.

The format of the configuration file mirrors that of a Windows .ini file and is defined as follows:

- Section names are contained in square brackets e.g. [SectionName].
- A section ends when another section begins or at End Of File (EOF).
- Comments are lines beginning with a semicolon (';').
- All text following a comment is ignored until the line is terminated.
- Keys are of the form keyname=value where keyname is a unique identifier within the section and value is the value of the specified key.
- Section names, key names and key values must not contain white space.
- Section names and key names are case sensitive.

The functionality in the Web Server plug-in is controlled using specific named sections in the configuration file. The omnissrv.ini file can contain the AllowConnectionsTo section which controls access to the Omnis App Server. The .ini file can also include either a DefaultConnection or OverrideConnection section (but not both), which either provide default parameters for the Omnis App Server or override parameters posted to the Omnis App Server from the http web server.

Controlling Server Access

You can control access to your Omnis App Server by including the [AllowConnectionsTo] section in the configuration file. This section contains a list of key names of the form *address<n>* where *n* is a sequentially numbered character starting at 1. When this section is present, connections to Omnis App Servers are limited to those defined in the specified key values. In the event that the oserver parameter is defined as a port number, only the port number is required.

For example, in the following AllowConnectionsTo section connections are limited to Omnis App Servers running on the local machine on port 5920 and the remote machine 192.168.0.2 on port 5920.

```
[AllowConnectionsTo]
  address1=5920
  address2=192.168.0.2:5920
```

Note that the local IP address in the configuration file cannot be resolved. Imagine the server plug-in and the Omnis App Server are on the same machine, with an IP address of 192.168.0.3. If the incoming request was of the form data- OmnisServer=5920, the configuration file has to match this form. So if you want to allow only connections to port 5920, you would have to add this line to the [AllowConnectionsTo] section: address1=5920. If you use the expanded form of the address, i.e. address=192.168.0.3:5920, the server plug-in would deny the request. In the event of a denial of service the plug-in returns a HTTP 403 error with the following message '*Access to the resource has been denied*'.

Default Connections

You can provide default connection parameters in the [DefaultConnection] section of the configuration file. This section provides a means of adding missing values into an HTTP post, or in effect, providing a complete set of default parameters if none are provided in the HTTP post. When they are present in the HTTP request, the values in DefaultConnection are ignored and the values are taken from the original request. The DefaultConnection section can contain the following keys:

- OmnisServer
- OmnisClass
- OmnisLibrary
- PostDataParamName
- Any number of additional parameter pairs in the form *Parameter Name=value*

The OmnisServer, OmnisClass and OmnisLibrary mirror the operation of the identically named remote form parameters. The value of the PostDataParamName key specifies a variable name for all the content of the HTTP post. All other keys are assumed to be parameters. They are passed to the Omnis remote task and appear as columns in the row variable. The column name is the key name and the value matches the value of the key. One thing to note, if the parameter is present in the original request and the configuration file also contains a definition for the parameter, the value is always taken from the request even if the parameter has no associated value. For example:

```
[DefaultConnection]
  OmnisServer=192.168.0.1:5920
  OmnisClass=remoteTask
  OmnisLibrary=TEST
  param1=value1
  param2=value2
  PostDataParamName=PostData
```

In the context of the above DefaultConnection section, consider the following URL which attempts to connect to Omnis:

```
/omnis_apacheini?OmnisClass=remoteTask&param1=1234
```

The OmnisClass and param1 values are taken from the URL while the other values are taken from the DefaultConnection section. In this case, no OmnisServer and OmnisLibrary parameters are provided in the query string, so those values are taken from the configuration file. Therefore the plug-in will amend the query string to:

```
/omnis_apacheini?OmnisClass=remoteTask&OmnisServer=192.168.0.1:5920&OmnisLibrary=TEST&param1=1234&param2=value
```

Note PostData is empty as the content-type is application/x-www-form-urlencoded, so in this case the data is not passed to Omnis.

Overriding Connections

You can override the server parameters passed to the Omnis App Server by an HTTP post by including a [OverrideConnection] section in your configuration file. In this case, all the values in the request are ignored, and the Omnis App Server uses values from the configuration file. The OverrideConnection section may contain the following keys with associated values:

- OmnisServer
- OmnisClass
- OmnisLibrary
- PostDataParamName
- Any number of additional *Parameter Name=value*

These keys function exactly as described in the DefaultConnection section. An example OverrideConnection section is as follows:

```
[OverrideConnection]
  OmnisServer=192.168.0.1:5920
  OmnisClass=rtTest
  OmnisLibrary=TEST
  param1=value1
  param2=value2
  PostDataParamName=PostData
```

In the context of the above OverrideConnection section, consider the following URL which attempts to connect to Omnis:

```
/omnis_apacheini?OmnisClass=remoteTask&param1=1234
```

In this case, the values in OmnisClass and param1 submitted in the post are ignored, and all the values for the post are taken from the DefaultConnection section in the configuration file. Therefore the query string is amended to:

```
/omnis_apacheini?OmnisClass=rtTest&OmnisServer=192.168.0.1:5920&OmnisLibrary=TEST&param1=value1&param2=value2&
```

Note PostData is empty as the content-type is application/x-www-form-urlencoded, so in this case the data is not passed to Omnis.

Creating Standalone Mobile Apps

In addition to using the JavaScript Client in the web browser on any computer or mobile device, you can embed your JavaScript Client based remote forms into a Standalone app or "wrapper", which you can deploy to end users as a self-contained app which they can install onto a mobile device. To create a standalone app, we provide a *JavaScript Wrapper Application* for each supported mobile platform, which currently includes Android and iOS. The wrapper applications create a thin layer around a simple Web Viewer which can load the initial JavaScript remote form for your mobile application.

The wrapper applications allow you to deploy mobile apps that end users can run either in *Offline* or "serverless mode", without any connection to the Omnis App Server, or in *Online* mode which would allow end users to connect to the Omnis App Server to synchronize their data and update the application content.

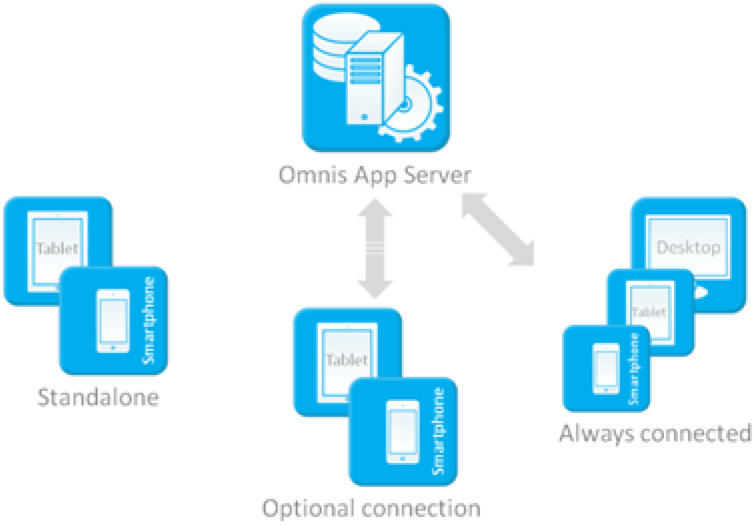


Figure 219:

- **Offline or Serverless mode**
  end users can run your app in standalone or "offline" mode without ever connecting to the Omnis App Server or a database server. The Application Files (remote form definitions, scripts, etc) are bundled with the wrapper app to create a single, clickable application file; this allows complete "offline" operation, or a one-off connection can be made to the Omnis App Server to install the applications files and from there on a connection to the Omnis App Server would not be needed
- **Optional connection**
  end users have the option to switch to "online" to synchronize the database and app content via the Omnis App Server; this mode would suit end users who have an intermittent connection, but often need to synchronize their data with a central location

Wrapper Application Source Files

The projects and source code for the wrapper applications are available to download from the Omnis website: https://omnis.net/developers/resources/download/jswrapper.jsp

The ZIP files contain template configuration files, together with the project and source files so you can build your standalone app or customize the wrappers if required.

There is a separate manual called 'Building The [Android / iOS] Wrapper' available on the Omnis website about using and customizing the wrapper applications for mobile app deployment: this manual contains all the latest information for each of the supported mobile platforms and will be updated regularly to keep abreast of any changes in the build process.

Configuring the Wrapper Application

The wrapper applications can be configured to run a single JavaScript remote form as the entry point to the app, which can be configured in a configuration file called **config.xml.**

Configuration file

The config.xml file contains the URL for the page containing your JavaScript remote form and depending on the platform, may contain a number of other parameters specific to your platform. The config.xml is standardized for all platforms, and is based on the following structure:

```
<?xml version="1.0" encoding="utf-8" ?>
<settings>
  <AppTitle>0</AppTitle>
  <AppStandardMenu>1</AppStandardMenu>
  <AppTimeout>300000</AppTimeout>
  <MenuIncludeSettings>1</MenuIncludeSettings>
  <MenuIncludeOffline>1</MenuIncludeOffline>
  <MenuIncludeAbout>1</MenuIncludeAbout>
  <SettingsFloatControls>0</SettingsFloatControls>
  <SettingsScaleForm>1</SettingsScaleForm>
  <SettingsAllowHScroll>0</SettingsAllowHScroll>
  <SettingsAllowVScroll>0</SettingsAllowVScroll>
  <SettingsMaintainAspectRatio>0</SettingsMaintainAspectRatio>
  <SettingsOnlineMode>1</SettingsOnlineMode>
  <ServerOmnisWebUrl>http://172.19.250.25:5911</ServerOmnisWebUrl>
  <ServerOnlineFormName>/jschtml/rfOnline</ServerOnlineFormName>
  <ServerOmnisServer></ServerOmnisServer>
  <ServerOmnisPlugin></ServerOmnisPlugin>
  <ServerOfflineFormName>rfOffline</ServerOfflineFormName>
  <ServerAppScafName>mylib</ServerAppScafName>
  <TestModeEnabled>0</TestModeEnabled>
  <TestModeServerAndPort>172.19.250.25:5911</TestModeServerAndPort>
</settings>
```

The config.xml contains the following properties (note that some may not be included on a particular platform):

- **AppTitle**
  whether or not the app displays a title bar at the top. Note that hiding the title on Android 3 will hide the ActionBar which will remove access the testing menu
- **AppStandardMenu**
  whether or not the standard menu is displayed at the top.
- **AppTimeout**
  the time in milliseconds after which the app will close after being sent to the background.

For all platforms:

- **MenuIncludeSettings**
  whether the "Settings" menu option is available in the app.
- **MenuIncludeOffline**
  whether the runtime menu option is available to switch to offline mode.
- **MenuIncludeAbout**
  whether the "About" menu option is available in the app.

The following settings control the scaling of the app (remote form) on the device:

- **SettingsFloatControls**
  when SettingsScaleForm is "0" (false), the client uses the $edgefloat property of each JavaScript Client control on the form. When applying the screen size, the client uses $edgefloat to float the edges of controls (note that the component values are not supported, just the edge-related values). If the form is wider or taller than the screen, floating only occurs if the relevant SettingsAllowHScroll or SettingsAllowVScroll parameter is false. The amount by which the controls float is the difference between the actual screen width or height and the designed width or height of the form for the closest matching layout breakpoint. The value of $edgefloat is stored for each layout brealpoint

- **SettingsScaleForm**
  If you set this to “1” (true), the client scales the form to fit the available screen space. The scaling factor is the screen width or height divided by the value of the closest matching layout breakpoint. For these purposes, the actual screen size excludes the operating system areas such as the status bar
- **SettingsAllowHScroll** and **SettingsAllowVScroll**
  set these to “1” if you want to allow horizontal or vertical scrolling of the form respectively, or “0” if not
- **SettingsMaintainAspectRatio**
  If you set this to “1”, scaling maintains the aspect ratio of the form. When turned on, and depending on SettingsAllowHScroll and SettingsAllowVScroll, it may reduce the scaling factor in one direction, to make the form fit, and center the form vertically or horizontally as required
- **SettingsOnlineMode**
  whether the app starts in Online mode.

The following settings relate to the Omnis App Server:

- **ServerOmnisWebUrl**
  URL to the Omnis App Server or Web Server. If using the Omnis App Server it should be http://<ipaddress>:<omnis port>. If using a web server it should be a URL to the root of your Web server. http://myserver.com
- **ServerOnlineFormName**
  route to the form’s .htm file from ServerOmnisWebUrl. So if you’re using the built in Omnis server, it will be of the form /jschtml/myform.htm. If you are using a web server, it will be the remainder of the URL to get to the form, e.g. /omnis-apps/myform. (Do not add the .htm extension!)

Only ServerOmnisWebUrl & ServerOnlineFormName are needed for Online forms. The other Server… properties are for Offline mode.

- **ServerOmnisServer**
  The Omnis App Server <IP Address>:<Port>.
- **ServerOmnisPlugin**
  If you are using a web server plug-in to talk to Omnis, the route to this from ServerOmnisWebUrl. E.g. /cgi-bin/omnisapi.dll
- **ServerOfflineFormName**
  Name of the offline form. (Do not add .htm extension!)
- **ServerAppScafName**
  Name of the App Scaf. This will be the same as your library name.

The remaining parameters refer to test mode.

- **TestModeEnabled**
  whether the app will start in test mode (Ctrl-M on form from Studio to test on device)
- **TestModeServerAndPort**
  the <ipaddress>:<port> of the Omnis Studio Dev version you wish to use test mode with.

You can also change these parameters by pressing the menu button on the mobile device, and using the menu options to change them. The app remembers the last setting made via the menu, so the config.xml lets you set the initial values in the wrapper application.

Access Permission Requests

When you deploy your app and the end user downloads it, the app must request permissions to access various areas of the device, for example, the app must request access to the device Contacts, Camera, or Location (GPS) if this functionality is required in your app.

It is considered bad practice and potentially confusing for the end user to include unnecessary permissions for your app, especially if you are distributing your app through one of the online app stores. When downloading/installing your app, the user can see which permissions your app has requested access to, so any unnecessary permission requests may give the user the impression that your app is malicious and they may not download and use your app.

Testing Remote Forms in a Wrapper App

During development, you can open a JavaScript remote form in a wrapper application using the *Test Form Mobile* (Ctrl-M) option, assuming a wrapper application is setup and enabled (otherwise you can still test your mobile forms in a desktop browser on your development computer before you setup the wrapper app). This option appears beneath the 'Test Form' option in the remote form context menu. The Test Form Mobile option is only displayed in the relevant menus when both:

- A wrapper application is enabled for test form (see the menu of the Android app, and the system settings for the iOS app)
- A wrapper application is connected to the Omnis App Server, using the test form parameters.

The $designshowmobiletitle property determines whether or not the title of a wrapper application is visible when you use the Test Form Mobile (Ctrl-M) option. For deployment, config.xml allows you to configure whether or not the title is displayed in the wrapper application.

Serverless Client

PLEASE NOTE: you will require a new alternative Development license to enable the $serverlessclient property in a remote form: the property will remain grayed out without this alternative license. (Also note that previous versions of the Serverless Client provided local database support using UltraLite and a MobiLink server from Sybase but this setup is no longer supported.)

You can switch a remote form to operate in "Serverless Client" mode in which case your mobile app can operate entirely without a connection to the Omnis App Server. In order for a remote form to be available in Serverless or offline mode, you must set its **$serverlessclient** property to kTrue.

Any mobile app containing remote forms in serverless client mode and you can switch between "offline" and "online" modes, if required. Separate forms can be used for the same app in offline/online modes to provide different functionality.

The Serverless Client also includes **Local Database** support, utilizing a SQLite database, and even provides the ability for your offline form to synchronize an end user's local database with an online database. This means that a user could switch to offline mode while in areas of no or patchy network coverage to continue working, and then switch back to online mode when back in the office.

How does it work?

The Serverless Client has provision for a local client-side database. This can be used as a local database for the standalone mode for your app, or it can be used to synchronize with an online "Consolidated Database" (CDB). In the latter case, the local database is used for storing tables held in the server-side database and for caching SQL transactions performed whilst the Omnis App Server is not available.

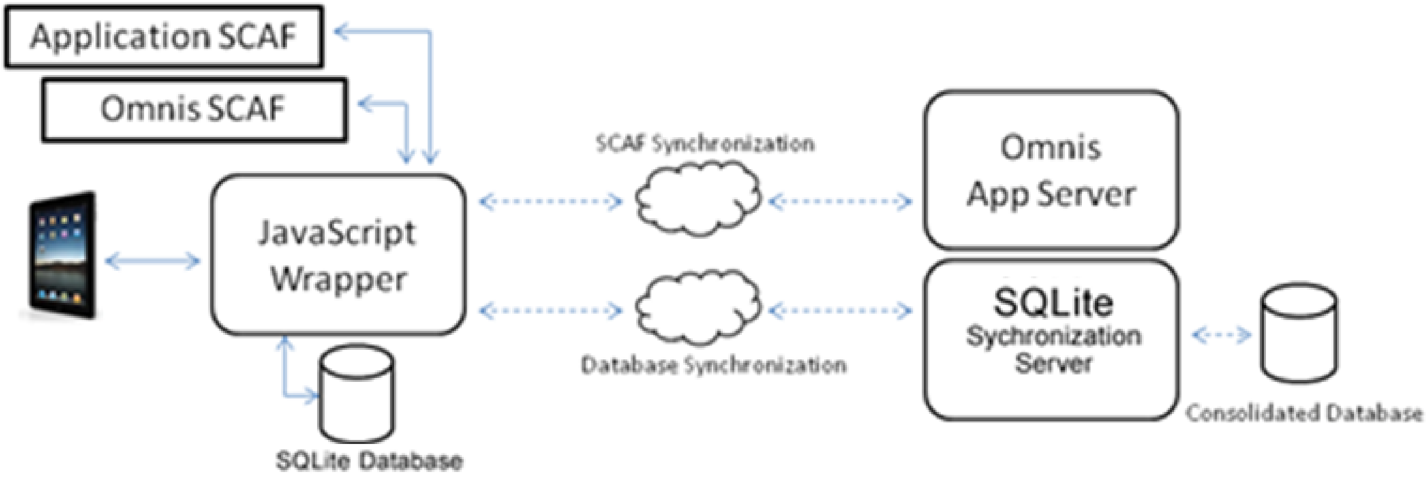


Figure 220:

JavaScript Serverless Client allows forms to work in 'offline' mode

The client-side database can be SQLite and the synchronization of the local database with a "consolidated database" administered by the SQLite Synchronization Server provided by Omnis Software.

You can download the 'SQLite Synchronization Server' manual from the Omnis website: www.omnis.net/download

Serverless Client Methods

All methods running in a Remote form in serverless client mode *must be set* to 'Execute on Client' (right click the method name in the method editor and select 'Execute on Client'). From Studio 10.0.1 onwards the default execution type for new methods added to a serverless client remote form is client-executed.

Initialization and Termination Methods

The $construct() and $destruct() methods in a remote form cannot be executed as client methods, therefore you can create methods with the names $init() and $term() which perform a similar function that can be executed on the client. The $init() and $term() methods can be used in standalone apps running inside the JavaScript Client wrapper application in which all methods must be executed on the client.

The $init() method is called after the form and the client script files have been loaded. This allows you to do any final initialization of the remote form. The $term() method is called when a remote form instance destructs.

You can use $cinst.$layouttype in client methods, including $init(), to get the current layout breakpoint of the remote form on the client (you can use $cinst.$screensize for the old $screensize based forms).

Remote Task Instances

The Application Wrappers send a unique device ID when connecting to the Omnis App Server. Omnis checks whether there is already a remote task instance with the same device ID and form name in the current library, and if it finds one, it will close it before opening a new task instance. This means that wrappers will not free the remote task connection when they timeout (as you cannot trap this event), but when the app is re-opened, it will close the old task before opening a new one.

Serverless Client Application File (SCAF)

The *Serverless Client Application File* (SCAF) is a SQLite database that contains all of the resources necessary for a mobile application to run locally in the wrapper in standalone mode. These resources include JavaScript scripts, CSS files, image files and Omnis remote forms. There are two SCAF files needed for each wrapper application:

- **Omnis SCAF** (omnis.db)
  files needed to run the JavaScript Client
- **Application SCAF** (<library_name>.db)
  contains all your application files

Omnis Studio will generate these SCAF files automatically in the '/html/sc' folder under the main Omnis folder. These files need to be placed on the Omnis App Server in the same location for end users (mobile clients) to access if necessary (see below).

Whenever you save a remote form which has $serverlessclient set to kTrue, it will update the application files in your Omnis folder. A message is displayed while Omnis exports all the necessary files.

If you need to rename your library, rename the library file itself in the file browser; you cannot rename it by changing the $name property as it will revert to the original name when the library is re-opened. If you change the $defaultname library property, the new name is used in the SCAF which is then rebuilt the next time the library is opened.

Updating the SCAF

When you make changes to CSS files or scripts you must update the SCAF to ensure your app is built using an up-to-date SCAF. You can update the SCAF in the Studio Browser by clicking on the 'Omnis Studio' node and selecting the 'Update Omnis SCAF' option. For example, the SCAF needs to be updated after the 'user.css' has been changed to ensure the omnis.scaf contains the updated style sheet. Quitting and restarting Omnis Studio also updates the SCAF.

Deployment of SCAF Files

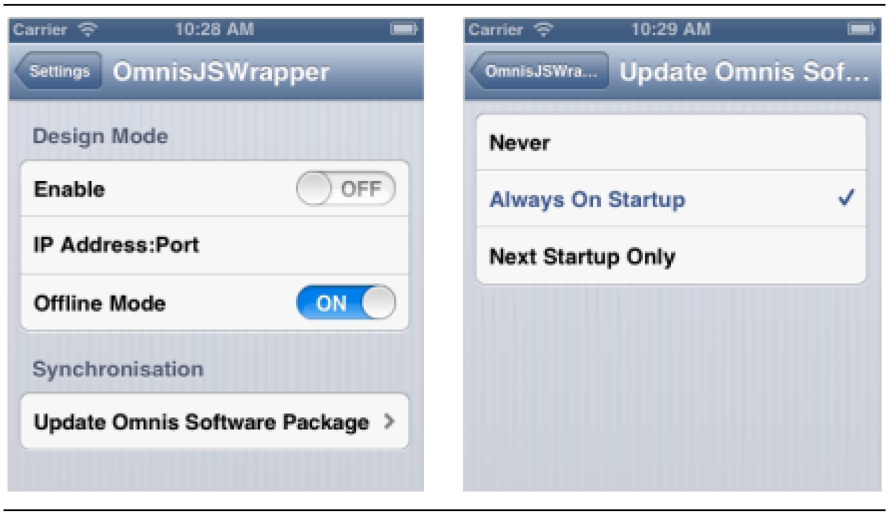
When the wrapper application is executed for the first time in offline mode it will check if any SCAF files are bundled with the application (see the wrapper building section for info on how to do this). If these files exist, they are copied into the application space on the device and used in the wrapper. Note that bundling SCAF files with the application will increase the size of the application bundle.

If SCAF files are not distributed with the application the application will attempt to connect to the Omnis App Server on startup and download the latest versions of the SCAF files.

For the application to know which application SCAF to use the option: <APPSCAF>name</APPSCAF> (where name is the name of the Omnis library)
is used in the config.xml file within the wrapper application.

Updating SCAF Files

Once the app has been installed onto a device, it will add an entry to the 'Settings' app of the device, with the same name as your app.



Design Mode

Enable OFF

IP Address:Port

Offline Mode ON

Synchronisation

Update Omnis Software Package >

Never

Always On Startup ✓

Next Startup Only

The wrapper configuration sub menu "Update Omnis Software Package" contains options for updating the SCAF files. The available options are:

- **Never**
  the SCAF will never be updated
- **Always On Startup**
  will attempt to connect to the Omnis App Server and update the SCAF files every time the application is run
- **Next Startup Only**
  will attempt this update only on the next execution of the wrapper application

It is also possible for the end-user to update the SCAF from inside the app by swiping down the screen to open the runtime menu and selecting the appropriate menu item.

Database Support

The JavaScript wrapper application contains embedded support for SQLite and provides client-executed methods with access to a local private SQL database. The database can currently only be used by Serverless Client applications in offline mode, inside the wrapper application on Android and iOS.

All interactions between the JavaScript Client and the wrapper will be asynchronous, so the database API also takes this into account.

Schema and Query Classes

The JavaScript Client-executed method code generator restricts the $definefromsqlclass() method for use with either a query or a schema class name as the first argument although it is still possible to pass a subset of column names required using parameter two onwards. This will allow for example:

```
Do list.$definefromsqlclass('SchemaName')
```

and the code generator will expand this into JavaScript to define the list or row with the columns from the schema.

Data Types

When creating rows to be used for bind variables, it is important that the data types of the columns in the row match those in the database.

Client methods only provide a 'var' data type when creating variables, which will generally be interpreted as Character type. As such, it is safest to manually add columns to your row, using the function:

```
Do lRow.$cols.$add(<name>,<data type>,<data subtype>,[<length>])
```

For example:

```
Do lRow.$cols.$add('Age',kInteger,kShortint)
```

The SQL Object

A Serverless Client application gains access to the embedded database using a SQL Object, which in this case is a property of the current remote form called $sqlobject:

```
$cinst.$sqlobject
```

For example:

```
Calculate oVar as $cinst.$sqlobject
```

All requests to the SQL object are asynchronous (except $getlasterrortext and $getlasterrorcode), and call a client-executed completion method ($sqldone) in the current remote form instance upon completion. Each request returns an identifier when called and the same identifier is passed as a parameter to the completion method, allowing the request to be identified. Thus, multiple requests may be in progress "simultaneously", although they will only execute serially in the wrapper.

Note that you do not need to provide a $sqldone() method although if you do not, errors may be ignored. On success, the returned unique identifier is positive. A negative value indicates an error code.

In the following sections, oSQL is a Var containing the SQL object returned by $cinst.$sqlobject.

$getlasterrortext()

```
Do oSQL.$getlasterrortext() Returns lErrText
```

Returns the error text of the last operation. "OK" implies success.

$getlasterrorcode()

```
Do oSQL.$getlasterrorcode() Returns lErrCode
```

Returns the error code of the last operation. 0 implies success.

$selectfetch()

```
Do oSQL.$selectfetch(cSQL, lBindVars, iFetchCap) Returns id
```

Executes a statement with a result set (typically select or select distinct) and fetches the initial set of rows.

- cSQL
  is the statement. This may be hand-coded, or the result of $select/$selectdistinct for a schema or query class. cSQL can contain bind variable place-holders in the form @[column_name], where column_name is the name of a column in lBindVars.
- lBindVars
  is a row variable referenced by one or more bind variable markers in the SQL text.
- iFetchCap
  is the number of rows to initially fetch (this can be kFetchAll to fetch all rows in the result set).

For all object methods, note that lBindVars may contain columns not referenced by the SQL text. Only those columns referred to by name in the bind place holders will be read.
On completion, $sqldone() is called with the following parameters:

- The request id (as returned by $selectfetch)
- A list containing zero or more rows from the initial result set.

At this point it is your responsibility to copy or populate the appropriate form controls to display the data.

Example1:

```
Do iList.$definefromsqlclass('myQuery')
Do oSQL.$selectfetch($clib.$queries.myQuery.$select, iList, 100) Returns id
```

Example2:

```
Do oSQL.$selectfetch('select * from Table1 where age=@[age]', lBindVars,100) Returns id
```

$fetch()

```
Do oSQL.$fetch(selectfetchid, iFetchCap) Returns id
```

Fetches more rows from the result set generated by the last $selectfetch() executed.

- selectfetchid is the id returned by $selectfetch() and passed to $fetch(); a new ID is returned.
- iFetchCap is the number of rows to fetch.

In this case, the id returned by the call to $fetch() is a new ID. On completion, $sqldone() is called with the following parameters:

- The request id (as returned by $selectfetch)
- A list containing zero or more further rows from the result set.

$insert()

```
Do oSQL.$insert(cSQL, list) Returns id
```

Inserts one or more rows into a database table.

- cSQL is the insert statement. This may be hand-coded, or the result of $insert for a schema class. cSQL can contain bind variable place-holders in the form @[$column_name], where column_name is the name of a column in listorrow.
- listorrow is the list or row containing the data to insert.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $insert()).

Example:

```
Do oSql.$insert("INSERT INTO Product (name, quantity) VALUES (@[colName],@[colQuant])",lBindVars) Returns IDin
```

$delete()

```
Do oSQL.$delete(cSQL, row) Returns id
```

Deletes zero or more rows from a database table.

- cSQL is the delete statement. This may be hand-coded, or the result of $delete() for a schema class. cSQL can contain bind variable place-holders in the form @[column_name], where column_name is the name of a column in row.
- row contains the values referenced by the bind variable place-holders.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $delete()).

$update()

```
Do oSQL.$update(cSQL, newRow, oldRow) Returns id
```

Updates zero or more rows of a database table.

- cSQL is the update statement. This may be hand-coded, or the result of $update for a schema class. cSQL can contain bind variable place-holders in the form @[$column_name], where column_name is the name of a column in newRow or oldRow. If the bind variable is used in the SET clause, it will come from the newRow variable, if it is used in the WHERE clause, it will come from the oldRow variable.
- newRow is the row containing the values referenced by the bind variable place-holders of the new values, i.e. those specified in the SET clause.
- oldRow is the row containing the values referenced by the bind variable place-holders of the old values, i.e. those specified in the WHERE clause.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $update()).

$execute()

```
Do oSQL.$execute(cSQL) Returns id
```

Executes a SQL statement that does not return a result set, intended for use with DDL administrative commands such as CREATE, DROP and ALTER.

- cSQL is the SQL statement to be executed. Note that bind variable place holders are not supported.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $execute()).

The following methods may be used to obtain database meta-data:

$selecttables()

```
Do oSQL.$selecttables() Returns id
```

Retrieves table names defined in the local database.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $selecttables()).
- Single-column list containing the **TableName** of each table in the local database.

$selectcolumns()

```
Do oSQL.$selectcolumns(tableName) Returns id
```

Retrieves column names and type information for the specified table.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $selectcolumns ()).
- A list describing the table column definitions, defined with the following columns: **ColumnName** - name of the table column. **SqlType** - name corresponding to the column's SQL data type. **ColumnSize** - the size of a variable-length data type, e.g. for CHAR and BINARY. **Precision** - the numeric precision for a NUMERIC column. Zero for others. **Scale** - the numeric scale for a NUMERIC column. Zero for others. **Default** - the default value that was assigned to the column when the table was created.

$selectindexes()

```
Do oSQL.$selectindexes(tableName) Returns id
```

Retrieves column index information for the specified table.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $selectindexes ()).
- A list describing the table column definitions, defined with the following columns: **IndexName** - name of the index. **ColumnNames** - comma-separated list of column names used by the index. **PrimaryKey** - kTrue if the index was created with the PRIMARY KEY clause. **Unique** - kTrue if the index was created with the UNIQUE clause.

$sqldone method

The $sqldone method is the client-executed completion method for SQL objects. When you add the $sqldone method to a remote form Omnis adds pre-defined or boilerplate code, as well as the required parameter variables, and sets the method to execute on the client automatically. This saves you having to add the same code every time you want to create the $sqldone method – you can then add to or amend the code as you wish. The code added to $sqldone is:

```
# parameter vars pRequestId (Var type), pList (List) created
# local vars lErrorCode and lErrorText created
# Check for an error:
Do $cinst.$sqlobject.$getlasterrorcode() Returns lErrorCode
If lErrorCode<>0    ## sql error occurred, show message
  Do $cinst.$sqlobject.$getlasterrortext() Returns lErrorText
  Do $cinst.$showmessage(lErrorText,'SQL Error')
  Quit method
End If
Switch pRequestId
  # Add cases for the IDs returned by your requests here.
End Switch
```

The code first checks if there was an error, then creates a Switch statement to handle the results based on the request in pRequestId. If you do not want to use this code, just select the lines of code and delete them.

SQLite Database Support

The SQLite offline storage and synchronization process uses a SQLite database on the remote client device. More specifically, SQLite synchronization relies on SQLite databases on the server and on each client device to store user tables as well as synchronization status info. The SQLite Synchronization Server uses these tables to pass data to/from each synchronization client and to forward synchronization requests on to the Consolidated Database (CDB). The SQLite Synchronization process is described in the 'SQLite Synchronization Server' manual which you can download from the Omnis website: www.omnis.net/download

To use the SQLite database object, the mobile device application is linked with the *dbSQLite* library. The SQLite initialization parameters are as follows:

$syncinit()

```
Do oSQL.$syncinit(syncParams) Returns id
```

The SQLite module currently recognizes the following parameters:

Username – The synchronization user name (defined at the synchronization server).
Password – The synchronization user password (defined at the synchronization server).
HostString – RESTful connection URL to Omnis Sync Server.
Timeout – The timeout in seconds for synchronization operations.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $syncinit()).

Example:

```
Do con.$define(Username, Password, HostString, Timeout)
# define using local variables
Do con.$assigncols('user1','xxxxxx','http://192.168.0.10:7001', 5)
Do oSQL.$syncinit(con) Returns id
```

HostString

The HostString parameter is a RESTful connection URL to Omnis Sync Server. For a direct connection to the built-in Omnis server, the HostString should be of the form:

```
http://<ip-address>:<$serverport>
```

If you are connecting through a web server, you need to add the omnisrest… server plugin to your web server (in the same way as the other server plugins), and connect through that. The HostString should then be of the form:

```
http://<web server address>/<Omnis rest plugin>/ws/<XXX>
```

where <XXX> is either:

```
<Omnis $serverport> (if Omnis is on the same machine as the web server)
```

```
<Omnis server ip-address>_<Omnis $serverport>
```

```
<Server Pool>,<Omnis server ip-address>_<Omnis $serverport>
```

For example:

```
http://mysite.com/cgi-bin/omnisrestisapi.dll/ws/192.168.1.14_7001
```

or

```
http://mysite.com/cgi-bin/omnisrestisapi.dll/ws/POOL,7001...
```

$sync()

This method invokes a request for uplink synchronization followed by downlink synchronization. Only tables previously configured for uplink (or normal) synchronization will upload IUD requests to the SyncServer. Likewise only tables configured for downlink or (normal) synchronization will receive IUD requests.

Please refer to the 'SQLite Synchronization Server' manual for information on the design, implementation and usage of the synchronization server. You can download this manual from the Omnis website: www.omnis.net/documentation

No Database Support

If the remote client application does not require database support, the application can instead be linked with the *dbNoSQL* library. This library provides stub definitions for the database API calls required by the wrapper application. Note that in this mode however, any calls to the SQL object will fail.

JavaScript Client Wrapper Application

The wrapper encapsulates a *WebView* which hosts the JavaScript Client application. The wrapper initializes using the supplied config.xml file which also informs the wrapper of the HTML page to load for the application.

The wrapper can be configured to connect to the Omnis IDE as a client for testing purposes. This is achieved via the Test Mobile Form menu option in the Studio IDE. The wrapper also provides device-independent access to features such as GPS, the camera and audio interface using the **Device Control** (see the *JavaScript Components* chapter for details about accessing device features).

To support Serverless operation, the WebView runs local scripts that contain client executed methods.

Before you compile the app, you will need to customize the *config.xml* file.

When the client operates in online mode it uses the URL parameter from the config file to load the remote form and the wrapper behaves like a standard JavaScript application. If the client is to run offline however, the other config parameters are used to allow the wrapper to (optionally) update its local copy of the application, or to run the forms locally.

iOS Wrapper Project

The iOS wrapper project has two targets, both with differing local database support, and one with support for Push Notifications. These are:

- **OmnisJSWrapper** – includes embedded SQLite database which is used for local database support, plus Synchronization with a back-end server using the SyncServer.
- **OmnisJSWrapper** – as above but with Push Notifications.

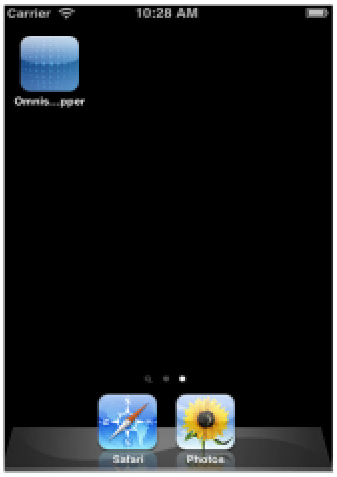


Figure 221:

iOS Wrapper Licensing

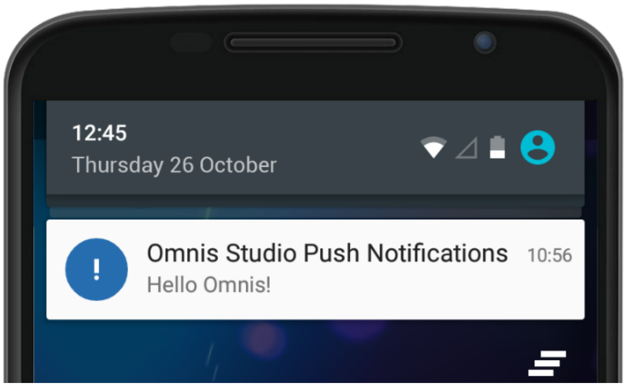
The iOS application wrapper uses the UICKeyChainStore wrapper, created by kishikawakatsumi and governed by the MIT license: more info is available here:
https://github.com/kishikawakatsumi/UICKeyChainStore

If you distribute your Omnis app using the iOS wrapper you will need to comply with the terms of this license and include the MIT requirements in your own software license.

Push Notifications

Push Notifications are supported in the Android and iOS JavaScript Wrappers (version 2.0+) which means you can send messages to any clients that have your mobile app installed (even if it is not running). In this respect, the ability to send push notifications provides a powerful and interactive feature that proactively encourages end users to open and use your mobile app.

A notification or message pushed to a client could include an important news item, a message to users about a new entry into the database, or anything else you want your end users to know about. You can include a payload of data to send with the notification, which will be passed to your Remote Form, allowing you to provide a response or outcome to the user clicking on the notification. The following is a notification on an Android phone.



12:45
Thursday 26 October

Omnis Studio Push Notifications 10:56
Hello Omnis!

Figure 222:

Setting up Push Notifications

Support for notifications is provided via the *Cloud Messaging* or *Push Notification Service* on the respective platform, which must be enabled in your mobile app project when it is built using the latest JavaScript Wrapper SDK. To setup notifications in your app on Android and iOS, you will need to use *Firebase* from Google.

In order to manage notifications, it is possible to create groups of devices, and send notifications to particular groups, or individual devices. All functionality can be achieved in your Omnis code (using the notation), or using an admin tool, called **Push Notifications,** under the Tools menu on the Omnis menubar. Note the tool is an Omnis library located in the Startup folder which must be present for Push Notifications to work in your mobile apps, including your Omnis code, and for the Omnis App Server configuration to be setup.

For further information about setting up Push Notifications in your mobile apps, see the Push Notifications document on the JS Wrapper Download page.

Omnis App Manager

The **Omnis App Manager** is an iOS app that allows you to connect to and test your Omnis mobile applications, and manage multiple such configurations – it also allows you to test your iOS apps without having to go through the submission process with Apple which you will need to do when you want to deploy your completed iOS app. Therefore, the Omnis App Manager should be used primarily for testing, and when you are ready to deploy your Omnis application, you can download the JavaScript Wrappers from the Omnis website, and build a completed stand-alone app.

Specifically, you can use the App Manager to test functionality in the JS **Device** control before compiling your complete app using the wrappers. You should note that the App Manager does not support Push Notifications, so you will need to compile your app using the wrappers to test notifications.

You can download the Omnis App Manager from the Apple Appstore using the following shortcut: https://bit.ly/omnis-app-manager

There is no app manager for Android since you can easily build the Android wrapper on any platform and sideload the app to devices to test it, without having to go through the App Review process.

Creating an App Configuration

To set up the App Manager to test your mobile app, open the app manager, and click on the + icon to add a new configuration. To test an app you can just fill out the first three fields, as follows:

- **Name**
  a suitable display name for the app.
- **Host Server**
  the IP address of your development computer or Omnis Server, including the port number, e.g. 9814. The server address should be something like: http://192.168.1.100:9814. The port number is stored in the $serverport property under the Prefs option in the Studio Browser. If you are using a web server, this should be the address of your web server.
- **Online Form URL**
  the location and name of your test form, such as /jschtml/<remote-form-name>.htm. Note that you need to have tested your remote form in order to open the test HTML page which is created automatically when you open a remote form in design mode. If you are using a web server, this should be the path to your form's .htm file on your web server, relative to 'Host Server' above.

When the app information is complete tap on Save. To open the app (form), tap on the app name. Your remote form will open in the app manager allowing you to test its functionality, for example, you could test actions in the Device control.

The remainder of the settings in the app configuration are for setting up a standalone application running in offline mode as follows:

- **Offline Form Name**
  the name of the remote form used for offline mode (the $serverlessclient property of the form is set to kTrue), minus the .htm extension, e.g. jsOffline.
- **App SCAF Name**
  the name of the SCAF file for the application, generally a lower-case version of the library name, e.g. myapp.
- **Web Server Plugin**
  if you are using a web server plug-in to talk to Omnis, this is the name and location of the plug-in from the server named in the 'Host Server' field, e.g. /cgi-bin/omnisapi.dll.
- **Omnis Server**
  This is only necessary if you are using a web server with the Omnis web server plug-in, and can be either: 1) the Omnis port number (if Omnis is running on the same computer as the web server), or 2) <ip-address>:<port>, e.g. 192.168.1.100:9814.

- **Omnis Studio Version**
  the version number of Omnis Studio, e.g. 10.2
- **Start Offline**
  enable to start your app offline
- **Disable UTC Data Conversion**
  disables the automatic conversion of datetimes to UTC from local time when they are sent to the server.

Next are the database settings.

- **Local Database Name**
  this is the name of the local SQLite database to be used, including the .db extension, e.g. local.db. Multiple configurations with the same name here will share the same local database.

The Behaviour settings.

- **Disable Swipe To App List**
  this option disables the ability to swipe from the edge of the screen to open the apps list.

You can create multiple configurations in the app manager to test your Omnis mobile applications, but in order to test an app it needs to be open and running either in your development copy of Omnis or on the Omnis Server.

Headless Omnis Server

There is a "headless" version of the Omnis App Server on Linux that allows you to run your JavaScript Client-based web and mobile applications in a headless environment. The headless server is available for Linux only.

A so-called headless Omnis Server installed under Linux does not have a window-based interface, but can be controlled remotely from the command line in a Terminal window on the Linux machine, or you can configure the headless server using an Admin Tool which is accessible using an HTML page located in the HTML folder in the main Studio folder (the admin tool library is located in the Startup folder of the Headless Omnis Server tree).

Console Commands

The 'headlessAcceptConsoleCommands' item in the 'server' section of config.json controls whether or not the headless server provides a basic command line interface when used in a terminal window. The default setting for 'headlessAcceptConsoleCommands' is false. If set to True all Console Commands are recorded which means that 100% of CPU is used when the Headless server is run as a service: you will need to enable this option in config.json to accept all Console Commands.

Functions

The function *isheadless()* returns true when running in the headless server.

sys(231) returns zero in headless server.

sys(233) returns empty in headless server; it returns the title of the main Omnis application window in the full server.

Java

You can start the JVM at startup by setting the 'startjvm' in the 'java' section of config.json to true: it cannot be started by any other mechanism on the headless server.

Class Notation

If your Omnis code creates new classes using notation, there is a mechanism to initialise new objects using template files, located in the 'componenttemplates' folder in the 'Studio' folder. The folders are: componenttemplates/window, componenttemplates/remoteform, componenttemplates/report containing the template files to create window, remote form, and report instances, respectively. Each template file name is complibrary_compcontrol.json, with spaces converted to _ (underscore): it is a copy of an object.json file where only the properties and multivalueproperties members are used. complibrary and compcontrol are the component library and control name.

Restrictions

There are various restrictions or differences from full Omnis Server, as follows:

- Printing images to PDF in the headless server is restricted to PNG images (or true-color shared pictures) only.
- There is no port support.
- You should use the 'start' entry in the 'server' section of config.json to start the multi-threaded server
- The *Test if running in background* command always sets flag to true in the headless server.
- Several commands and notation methods generate an error if executed in the headless server e.g. open window, $open for a window, etc.
- Picture conversion functions are not supported: pictconvto, pictconvfrom, pictconvtypes, pictformat, pictsize (a runtime error is generated).
- Standard messages generated by the server (OK messages and errors) are sent to the server log file, or could be routed to the Terminal if appropriate

Printing JPGs

In order to print JPEGs from an application running on the Headless Linux Server, the ImageMagick package has to be installed.

Logging External Errors

The Headless Server logs a message when an external or external component cannot be loaded. This is a message of type headlesserror, and includes the system error text reporting the missing dependency that caused the component not to open.

Installing the Headless Server (Linux)

Download the Headless Omnis Server installer for Linux from: https://www.omnis.net/developers/resources/download/index.jsp

This install assumes you are running as Root or using sudo.

Update your version of Linux using the commands below that correspond to your distribution of linux:

```
Centos/Redhat: sudo yum update
Suse: sudo zypper update
Ubuntu/Debian: sudo apt-get update
```

Once updated, you will need to install the dependencies that Omnis requires to run, which are as follows:

```
Centos/Redhat: cups, pango
Suse/Debian: Runs out of box
Ubuntu: cups, libpango1.0
```

Once these are installed you can start the installer:

```
./Omnis-Headless-App-Server-11-x64.run
```

Follow through the installer as you would a normal install of Omnis Studio making sure your serial number is correct or the install will fail.

You can serialize the Headless Omnis Server using the OMNIS_SERIAL environment variable. If the Headless Server checks for serial.txt and there is no serial number saved in the omnis.cfg, it reads the serial number from the OMNIS_SERIAL environment variable before failing.

For Centos 7 and Redhat the service will not automatically start after a reboot, therefore you will need to manually add Omnis (or whatever you called your service) to the service autostart list using the following lines:

```
sudo /sbin/chkconfig --add homnis
sudo /sbin/chkconfig --list homnis (This line is to show that you have added homnis correctly)
sudo /sbin/chkconfig homnis on
```

You can now configure the Headless server using the Admin tool, as below.

To summarize the steps for each platform:

CENTOS7 & REDHAT

Required commands for Omnis to run on Centos:

```
sudo yum update
sudo yum install cups pango
sudo /sbin/chkconfig --add homnis
sudo /sbin/chkconfig --list homnis
sudo /sbin/chkconfig homnis on
```

SUSE

The Headless Server should work out of the box on SUSE, but we would recommend an update just in case:

```
sudo zypper update
```

Ubuntu 16.04, 17.04 & DEBIAN 9

```
sudo apt-get update
sudo apt-get install unzip libpango1.0 cups
```

Headless Server Admin Tool

There is an **Admin Tool (OSAdmin)** that you can use to configure the Headless Omnis Server: the Admin tool is implemented as a remote form and can be loaded in a web browser by opening the web page called 'osadmin.htm', which is located in the 'html' folder of the Omnis Server tree (not the SDK). However, before you can open this page to configure your headless server, you will need to set edit the osadmin.htm file to specify the location of your headless server. You need to edit the "data-webserverurl" parameter (enter either URL, IP address or Service name, and Port number, e.g. http://192.1.1.68:5000), then move the file to a location that allows you to open the file in a web browser and has network access to the headless server (the Headless Omnis Server installer should prompt you to set these options, but you may also like to change them manually).

The Headless Server Admin tool has a number of tabs that let you view or configure the server **Activity, Logs, Settings,** and **Users.** When you first open the admin tool in your browser, you are requested to login: use the default username: **omnis**, password: **0mn1s** (first character is zero). After logging in, you can change the password for the default user, or create other users.

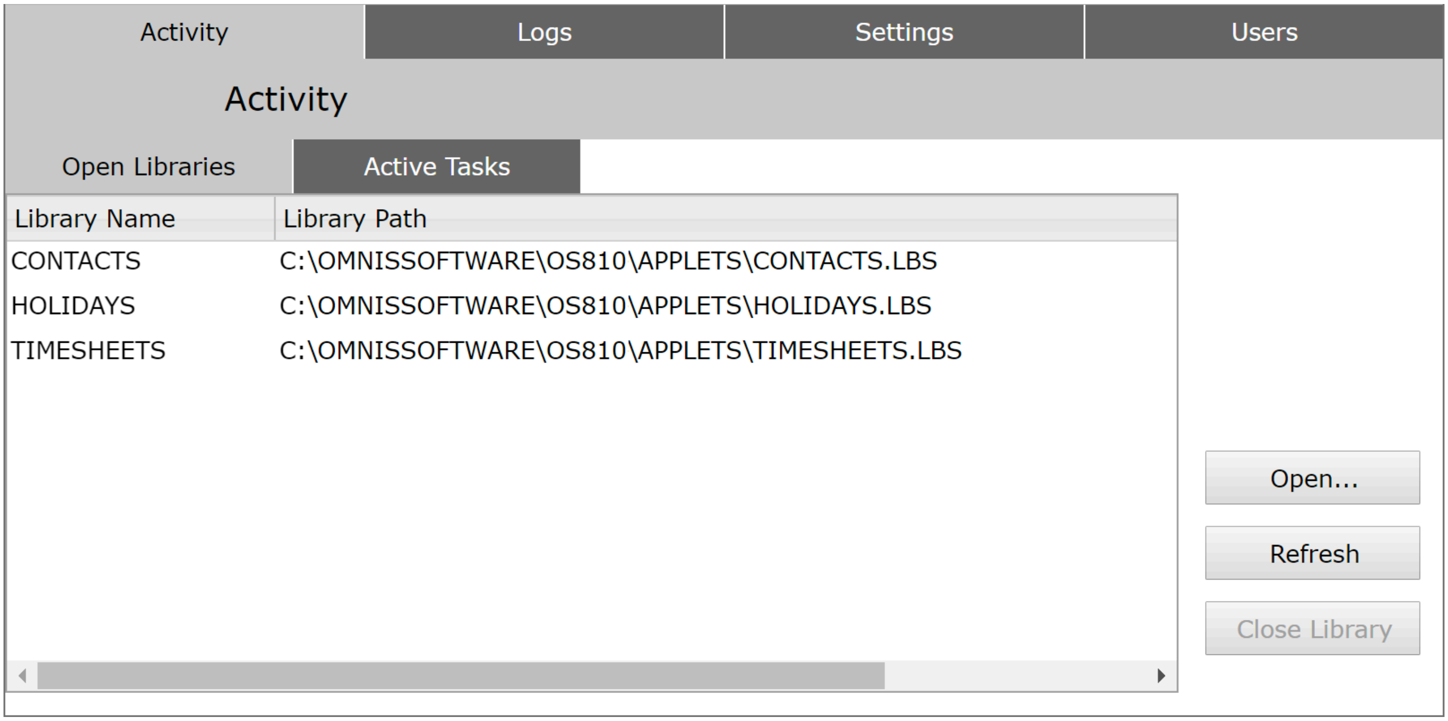
The configuration and settings for your Headless Server are stored in a SQLite database. The location of this database is specified in the "headlessDatabaseLocation" item in the "server" section of the Omnis Configuration file (config.json).

Activity

The **Activity** tab lets you see all **Open Libraries** on the server. You can use the Refresh button to refresh the list.

The **Open** button lets you open a library on the server; note the construct method will be run if present. You can click on a library in the list and close it using the **Close Library** button; note that closing a library will suspend all clients connected to that library.

The **Active Tasks** tab shows all current, active task instances or client connections on the server; you can select a task or connection and view its details. You can kill or close a task instance or connection using the Kill Task button; note that killing a task or connection will suspend the operation of the application for the connected client.



Library Name	Library Path
CONTACTS	C:\OMNISSOFTWARE\OS810\APPLETS\CONTACTS.LBS
HOLIDAYS	C:\OMNISSOFTWARE\OS810\APPLETS\HOLIDAYS.LBS
TIMESHEETS	C:\OMNISSOFTWARE\OS810\APPLETS\TIMESHEETS.LBS

Figure 223:

Logs

The **Logs** tab lets you view the logs for the Server:

- **Server**
  provides a log of the headless server activity (the location of the logs can be set under the main Settings tab)
- **Monitor**
  provides a log of all the active client connecttions (task instances)
- **Service**
  provides a log of all the errors or messages generated by the server including any messages in the trace log or information about any web service requests.

Under the Service tab, the **Configure** button lets you set up what messages are recorded in the log, including the attribute “folder” of “logToFile” which is the name of the path relative to the Omnis Server tree where the service logs are generated. These settings are added to the config.json for the server, under the “log” member:

```
"log": {
  "logcomp": "logToFile",
  "datatolog": [
    "restrequestheaders",
    "restrequestcontent",
    "restresponseheaders",
    "restresponsecontent",
    "tracelog",
    "seqnlog",
    "soapfault",
    "soaprequesturi",
    "soaprequest",
    "soapresponse",
    "cors",
    "headlessdebug",
    "headlesserror",
    "headlessmessage",
    "systemevent"
  ],
  "overrideWebServicesLog": true,
  "logToFile": {
    "stdout": false,
    "folder": "logs",
    "rollingcount": 10
  },
  "windowssystemdragdrop": true
}
```

Settings

Under the **Settings** tab you can specify the location of the Server and Monitor logs, plus the timer period and size of the logs. You can also set up the Server Port, number of Server Stacks, and the Timeslice for the Headless server (and specified in the config.json file), and you can restart the service from here.

The default service name of the Headless server is "homnis" which is specified in the "server" member of the config.json file:

```
  {
    "server": {
    …
    "service": "homnis"
```

You can upload libraries from the developer version to the startup folder of the Omnis Server, as well as move .htm files to the root of your web server (/var/www/html); plus you can remove files. Under the **Settings** section you can set the path to webroot and the web server handler, which default to /var/www/html and /omnis_apache respectively.

If you are running a different web server or web root directory, you need to modify these settings before uploading. For example, if you are using the Omnis built-in web server, you need to set the webroot path to '[path to Omnis read/write directory]/html' and the handler to '_PS_'.

Furthermore, osadmin will change the htm uploaded to use the specified web handler and the server port Omnis is currently bound to, therefore changing an htm before uploading it could break this functionality, so do not edit the htm file in this case.

Users

The **Users** tab lets you update users or create new ones. The default omnis user can be changed here. When checked, the **Re-start Option** will allow a user to restart the server.

MultiProcess Server

Under normal operation, the multi-threaded Omnis Server does not take full advantage of multi-core processors, because it uses a *time-slicing model* that single threads the execution of Omnis code in all situations, other than when some sort of external call (e.g. a DAM call) is in progress. The concept of the **MultiProcess Server** (MPS) for the Linux Headless server has been designed to eliminate this short-coming and deliver significant performance improvements in your applications, by using a *multiprocess* rather than *multithreaded* server model.

When using the MPS in the Omnis Linux Headless Server:

- There is a *single main server process* that receives requests from clients.
- There is a *separate child process for each client,* represented by a single remote task.

The main server process passes the request to a child process which executes the request. The child process then passes its response back to the main process, which then sends the response to the client.

Each child process is created using a forking system; however, the server is implemented so that when a child process becomes free (because its remote task destructs), it can be added to a pool of free child processes, ready to be associated with a new remote task. This greatly improves performance.

One of the main features of the MPS is that it can be plugged into an existing server configuration, and it will still work with the load sharing process; in other words, it still has exactly the same interface via its server port.

Another major advantage of using the MPS is that since execution is isolated to a single client per process, any problem in the child process (a crash perhaps), will only result in a single client receiving an error, and the server will continue running.

Configuration

To use the **MultiProcess Server** (MPS), you need to add some entries to the “server” section of the configuration file (config.json) for the *Linux Headless Server.* The entries are:

- **multiProcess**
  When multiProcess is true, the Linux Headless Server will start up in multiprocess mode; in this case, the entries “start” (Start server flag), “stacks” and “timeslice” in the server section of config.json are ignored as they are not relevant to the MPS.
- **maxChildProcesses**
  is the maximum number of child processes; default is 0 (zero), which means there is no maximum number of child processes
- **maxFreeChildProcesses**
  is the maximum number of free child processes (not associated with a remote task); default is 5

The new options are written to the “server” section of config.json like this:

```
"server": {
  "multiProcess": true,
  "maxChildProcesses": c,
  "maxFreeChildProcesses": f
},
```

The settings for maxChildProcesses and maxFreeChildProcesses depend on the size of the application, and the power and memory of the system, therefore a degree of system tuning will be required. Another factor to consider is the serial number. If the serial number only allows 100 users, then only 100 child processes can be created; however, if the serial number is a MAXW number, then 32000 users (the max number allowed) is probably too many for a single Omnis instance – in this case, you would want to prevent that many Omnis client connections coming to the single server, using load balancing.

When there is a free child process, a new client will connect a bit faster, as a free process is ready to go, so it is worth allowing these to build up to a sensible number.

In addition, you may want to handle timeouts for client connections. The headless server in all its variants (single-threaded, multi-threaded and multi-process) supports the following entries in the “server” section of config.json that provide some control over reading requests from a client:

```
"timeoutReads": true,
"readTimeout": s
```

These entries indicate if the server will timeout a connection from a client if the complete request is not received in “readTimeout” seconds.

Configuration files

Child processes never write to the files omnis.cfg and config.json.

Libraries

The MPS starts up just like the normal headless server. As such, it opens libraries in the startup folder, and constructs their startup tasks. There are however some rules that need to be followed, because of the way the forking process works:

- *No DAM connections can be left open* after the startup task constructors have run.

- *No files opened by FileOps or other externals should be left open* after the startup task constructors have run. This is because their file descriptors will be shared by each child process, because of how the forking process works.
- *A child process can only write to a library that it has opened or created itself* (this is opened with exclusive access by the child process). If the child process attempts to save a class to a library it did not open or create, Omnis ignores the error and returns success rather than an error code.
- *A child process can only close a library that it has opened or created itself*.
- *osadmin cannot open and close libraries in the MPS*.

Internally, when the forking occurs, the child process closes and re-opens the file descriptor (read-only, shared) for all open library files, since the otherwise shared file descriptor with the main process has a common shared file offset. Additionally, byte range locking calls in the child become no-ops.

Classes

As part of startup of the MPS, Omnis caches all class data from all open startup libraries in memory. This allows the class data to be immediately available to a child process after it is created using the forking process. As stated earlier, you cannot write to the startup libraries. Therefore, you *should not modify classes belonging to these libraries in a child process,* since the child process will typically be used for many remote task instances during its lifetime. However, this is not enforced.

Commands

You cannot use the following commands in the MPS:

- *Start server* and *Stop server*.
- *Set timer method* and *Clear timer method*.

All of the above generate the error 125446 (cannot use this command, function, or notation, in the multi-process server).

You cannot use the command *Quit Omnis* in a child process of the MPS. Attempting this generates the error 125437 (cannot use this command, function, or notation, when running in a child process in the multi-process server).

Finally, the commands *Begin critical block* and *End critical block* have no effect in the MPS. This is because each child process handles a single remote task.

sys() functions

There are two sys() functions to support the MPS:

- **sys(243)** returns true if and only if Omnis is running in MPS mode.
- **sys(242)** returns the child process ID, a character string that uniquely identifies the child process that is currently running. When the method is not running in the MPS, or not running in a child process, this has the value "0".

sys(242) can be used to identify the child process that is to process a request from a client: see the section "Using The Same Child Process" later in this document.

Process init method ($processinit)

When the MPS creates a new child process via the forking process, the child process runs the $processinit() custom method (if present) in the startup task of each open startup library. You can use $processinit() to carry out any initialization required to set up the environment in which all remote tasks handled by the child process will run.

Database Connections

Each child process *has its own SQL database connections*. You could use $processinit(), for example, to create a server pool containing a single database connection, that you can then use for all remote tasks that the child process handles.

Remote Task Methods

$maxusers

The MPS does not support the $maxusers property of a remote task.

$sendall()

Due to its multi-process architecture, the MPS does not support notation such as $iremotetasks.$sendall(), because if you execute this in a child process, it will only apply to the currently executing remote task.

To overcome this, Omnis includes (for all platforms, and all variants of server: single-threaded, multi-threaded and MPS) the following notation that allows you to *"broadcast"* a message to all remote tasks, including those running in another child process in the MPS.

Sending messages to Remote task instances using $broadcast()

There is a method of the $iremotetasks group of remote task instances called $broadcast() that can be used to send or "broadcast" a message via a public method to all task instances; its syntax is:

- **$broadcast()**
  $iremotetasks.$broadcast(cMethod, vListOrRow [, bWait=kTrue])

Calls the public method cMethod in all open remote task instances, passing vListOrRow as a parameter to each call. If bWait is kTrue returns a list of return values, containing a line for each remote task that executed the method.

cMethod does not need to exist in all remote tasks; if it does not exist, Omnis ignores the remote task.

When using bWait with value kTrue, all remote tasks must return the same data type. If the returned type is row, then the return value list is defined to have all of the columns of the row (so all remote tasks must use the same definition for their returned row), otherwise the return value list has a single column with the returned data type as its type.

Omnis Data Files

Omnis data files cannot be opened in the MPS. Attempting to open one causes the error 101437 (Cannot use data files (either because the serial number does not allow data files or because Omnis is running as a multi-process server or because Omnis was invoked with –runscript)).

Icon data files such as omnispic.df1 can still be used. As for libraries, the child process closes and re-opens (read-only, shared) the file descriptor for all open picture data files when it initializes itself after it has been created by the forking process.

Execution

When a new message arrives from a client, the main process inspects it. If the message is for an existing remote task, the main process sends it to the child process handling that remote task; to do this, the main process maintains a table that maps remote task connection id to child process. If the message requires a new remote task, then the main process either sends it to a free child process, or creates a new child process via the forking process (if configuration allows), and sends the request to the new child; if the configuration does not allow (the maxChildProcesses limit has been reached), the main process queues the request internally until a child process becomes available. This latter queueing behavior works best in a server handling RESTful, ultra-thin or SOAP web service requests, since these requests are usually processed relatively quickly; when the server is handling JavaScript client remote forms, it is best to allow essentially unlimited child processes, so a client can connect immediately.

Child processes send an event to the main process when their remote task destructs. The main process can then decide whether to tell the child process to exit (because the maximum number of free child processes has been reached), or add the child process to the pool of free child processes.

Licensing

Management of the server user count is handled using a shared memory semaphore. The main process initializes the semaphore with a count equal to the server user count. When a child process creates a remote task, and therefore needs a license, it waits on the semaphore. For a RESTful request, it waits indefinitely, and for other requests it tries to wait, and if the semaphore does not have availability, it rejects the request. This behavior is then equivalent to that of the multi-threaded and single-threaded servers, in that RESTful requests are queued waiting for a license, and other requests are rejected immediately if a license is not available.

When a child process deletes a remote task, it frees the license by posting the semaphore.

If a child process crashes while it is holding a license, the license will not be freed by the child. However, the main Omnis process attempts to restore the license semaphore count by posting to the semaphore appropriately, based on the server user count and the number of child processes currently assigned to a remote task.

Load Sharing

It is possible to use the MPS in conjunction with the Load Sharing Process, although this only really makes sense when each Omnis server accessed via the load sharing process is running on a separate machine, since the MPS is essentially providing a load sharing mechanism. The main process maintains the load sharing statistics and responds to the load sharing statistics request message.

Remote Debugging

Due to the dynamic nature of the Omnis environment, remote debugging is supported in the MPS only in the context of a single child process, explicitly created to execute clients that are to be remotely debugged – this is called a *Remote Debug Child*.

Debugging Startup

The following sections describe key points regarding how to remotely debug Omnis code running in the MPS.

If the member **pauseAtStartupUntilDebuggerClientStartsExecution** in the remote debug configuration is true, you can debug the startup tasks of the remotely debuggable code in the MPS. This code runs in the main process.

Start the MPS, use a Windows or macOS copy of Omnis to connect a remote debug client to the MPS, and debug the startup code.

After MPS startup completes, the remote debug connection closes.

Debugging the Remote Debug Child

After MPS startup, assuming remote debugging is enabled, the MPS creates the remote debug child, and the remote debug child then becomes the instance of Omnis visible to remote debug clients. Therefore, using the remote debug client on a macOS or Windows copy of Omnis, you can connect to the MPS remote debug child, and debug that directly.

If the member **pauseAtStartupUntilDebuggerClientStartsExecution** in the remote debug configuration is true, the remote debug child pauses, waiting for a remote debug client to connect, before running any $processinit() methods. In this case, the remote debug client has a hyperlink “Run Process Init” rather than “Run Startup”.

Making a Client Use the Remote Debug Child

For JavaScript remote form clients, specify ?omnisRemoteDebug=1 on the URL used to open the Omnis remote form.

For ultra-thin clients, include omnisRemoteDebug=1 in the query string used to invoke the ultra-thin request.

For RESTful clients, include the header omnisremotedebug in the request.

In all of the above cases, the request that requires the remote debug child will be queued if necessary, waiting for that child to become available.

Using the Same Child Process

The MPS allows you to cache data for ultra-thin and RESTful requests, to improve performance, using a query string parameter or HTTP header respectively, to specify the process ID (sys(242)) of the child process that is to ideally process the request.

When a request arrives that identifies a specific child process, the main process sends it to that child if the child still exists, otherwise it sends it to any available child.

For ultra-thin, the query string parameter is named ProcessID.

For RESTful, the HTTP header is named omnisprocessid. When using RESTful in conjunction with a process ID, you must always immediately respond to all requests, i.e. you cannot defer the response until later by assigning kFalse to $restfulapiwillclose.

Logging

The old-style web service logging to a data file is not supported for the MPS. Instead, you can configure the **datatolog** for the **logToFile** logcomp to include web services.

If you configure logging to go to standard output, by setting **stdout** to true in the logToFile object in config.json, logging from all processes in the MPS (main and child) will go directly to standard output, serialised between all of the processes using a shared mutex.

If you configure logging to go to a file in the logs folder, all child processes send their log data to the main process, which then writes the log data to file.

Command Line

All versions of Omnis on all platforms use –version as the switch to report the Omnis version and build number, rather than -version.

The Linux Headless Server has a number of command line switches. Pass the switch –help to display them all.

Command	Action
homnis –help	Print the help information and then exit
homnis –version	Print version and build number and then exit
homnis <sw>	Start the server using the start server and multi-process settings in config.json
homnis <sw> –st	Start the single threaded server ignoring the start server and multi-process settings in config.json
homnis <sw>–mt	Start the multi-threaded server ignoring the start server and multi-process settings in config.json
homnis <sw>–mp	Start the multi-process server ignoring the start server and multi-process settings in config.json
homnis <sw> –runscript=path <args>	Open the supplied library identified by <path>, construct its startup task, and pass the remaining command line arguments <args> to the $runscript method in the startup task

<sw> can be any combination of the following:

Switch	Meaning
–jscomments	Includes comments in client-executed JavaScript generate
–debugport=n	Overrides the configured remote debugging port
–pausestartup	Forces homnis to wait for a remote debugging client to cor running startup (and $processinit if relevant)
–debugscript	Starts the remote debug server at startup when invoked w
Any other user parameters that can be accessed from Omnis code using sys(202)	

–runscript

This mode allows you to use headless Omnis to run a script, that is, use headless Omnis to run some Omnis code within a shell script. For example, you could use the HTTP Worker Object in some Omnis code to invoke requests against an Omnis server. Used in conjunction with bash (or other shell) this can be quite powerful.

For example, you can write a script such as:

```
for i in {1..10}
  do homnis -runscript=mylib.lbs <args> &
done
wait
```

and execute it using

```
time ./myscript
```

This will create 10 instances of the run script homnis process, wait for them to complete, and report how long execution took.

In more detail, the path passed via the runscript argument is the path of an Omnis library, the startup task of which must contain a method named $runscript. This method receives as parameters the remaining parameters on the homnis command line, so for example you could pass a URL or an iteration count, or both. When homnis starts up in run script mode, it opens just this single library (ignoring startup libraries) and executes the method $runscript in the context of its startup task. The script library is responsible for calling *Quit Omnis* when it has finished. This allows it to start workers which may not complete until after $runscript has returned.

When homnis is running a script:

- It does not write to configuration files omnis.cfg and config.json.
- It does not accept command input from stdin.
- It only logs to stdout if logging is configured
- It will only open the script library passed as a parameter
- It will not open Omnis data files

You can use the printf() function to output a string from a script: printf(string[, newline=kTrue]) writes the string to standard output followed by a newline character if required (Ignored on Windows. Executes on macOS and Linux only).

External Components

It is possible that a loaded external component will not be in a good state after the forking process, probably due to problems with data structures in use by external libraries it is using (typically data structures containing some sort of operating system handle or file descriptor).

To cater for this, an external component with this kind of issue needs to return the flag EXT_FLAG_RELOAD_AFTER_FORK in the flags returned by ECM_CONNECT. This means that the main process unloads the component (calling ECM_DISCONNECT) after startup completes. Each child process then reloads the component (calling ECM_CONNECT again) as part of its initialization. As a result, each child process has a clean copy of the component.