What’s New in Omnis Studio 11

Omnis Software
March 2024
63-032024-01a

Table of Contents

About This Manual

This document describes the new features and enhancements in the latest revision of Omnis Studio 11, plus all previous revisions (35659, 35439) and the original Omnis Studio 11 release.

Please see the Readme.txt file for details of bug fixes and any release notes for Omnis Studio 11 Revision 35659.

NOTE: Where a new feature or an enhancement relates to an Enhancement Request or Customer reported fault, the fault reference is included to enable you to track your own ERs and reported faults.

Software Support, Compatibility and Conversion Issues

The following section contains issues regarding software support, compatibility and conversion in Omnis Studio 11 or above.

See the Readme.txt accompanying this release for information about faults fixed in this revision and for any last-minute release notes. See the Install.txt file to find out System Requirements for running the Development and Server versions of Omnis Studio.

Serial Numbers and Licensing

You will require a new serial number to run Omnis Studio 11 or above. Contact your local sales office to buy a license or obtain an upgrade serial number under your current support program; go to the Contacts page on the Omnis website: www.omnis.net

Library and Datafile Conversion

IMPORTANT NOTE:
See the section 'Class Locking and Library Conversion' (under the section about Revision 35659) about converting existing Omnis Studio 11 libraries in Omnis Studio 11 Revision 35659 (or above).

Converting 10.x Libraries

All Omnis Studio 10 or earlier libraries need to be converted to run in Omnis Studio 11. ONCE A STUDIO 10.0, 10.1 or 10.2 LIBRARY HAS BEEN OPENED WITH OMNIS STUDIO 11 IT CANNOT BE OPENED WITH STUDIO 10.x – THE CONVERSION PROCESS IS IRREVERSIBLE.

Converting 8.x or earlier Libraries

OMNIS STUDIO 11 WILL CONVERT EXISTING VERSION 8.1.X, 8.0.X, 6.1.X, 6.0.X AND 5.X LIBRARIES – THE CONVERSION PROCESS IS IRREVERSIBLE.

DISCLAIMER: *OMNIS SOFTWARE LTD. DISCLAIMS ANY RESPONSIBILITY FOR, OR LIABILITY RELATED TO, SOFTWARE OBTAINED THROUGH ANY CHANNEL. IN NO EVENT WILL OMNIS SOFTWARE BE LIABLE FOR ANY INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES HOWEVER THEY MAY ARISE AND EVEN IF WE HAVE BEEN PREVIOUSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.*

npm

npm 9.5.1 is now provided alongside Node.js. (Revision 35759, ST/EC/1755)

To launch npm you can run index.js inside the npm folder, e.g. ./node npm/index.js.

Microsoft SQL Server in the Community Edition

The ODBC DAM can now be used to connect to Microsoft SQL Server Express Edition in the Omnis Studio Community Edition. (Revision 35878, ST/*B/154)

Input Monitoring & Keystroke Receiving (macOS only)

In previous revisions of Omnis Studio 11, access to Chromium Safe Storage and Input Monitoring is requested on launch with no explanation. To address this, a new item called **monitorDockKeyEventsInfoPrompt** has been added to the Configuration file (config.json). (ST/IN/286)

When **monitorDockKeyEventsInfoPrompt** is set to true, Omnis provides a one-time prompt to give more information if Studio needs access to input monitoring; this is false by default. The text of the prompt can be altered by editing resource 19003. The config entry is automatically disabled once Omnis has been run.

More information is provided below (under Revision 35659), under the *Omnis Environment* and *oBrowser* sections.

Date and Time Conversion in the JavaScript Client

When converting Date and Time data in a SQLite database to dates on the JavaScript client, the subtype of any Date/Time data is now taken into consideration. (ST/*L/055)

Date/Time data fetched from a SQLite database were previously sent to the client as full datetimes. However, now '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

PDF Printing now uses **Node.js** to print reports, rather than Python (reportlab), as in previous versions. Note you do not need to adjust your application code or interface for this enhancement to take effect. (ST/EC/1635)

Since Python is no longer used for PDF printing, the python.zip file has been removed from the Omnis development tree.

Note: Node.js is used for PDF printing in Omnis subject to the MIT license from PDFKit: https://github.com/foliojs/pdfkit/blob/master/LICENSE

Oracle DAM

The Oracle8 DAM has been renamed to Oracle DAM. (ST/*O/201)

The DAM and session names ORACLE8DAM and ORACLE8SESS have been renamed to ORACLEDAM and ORACLESESS respectively. In addition, the Oracle DAM object "damora8" is now called "damoracle". The SQL Browser and VCS have been updated accordingly, as well as the oracledam.ini.

Omnis will map the old name including 8 to the new name without an 8 automatically, for example, when opening the Select Object dialog, and when creating the object.

Form and Report Object Limit

You cannot place an unlimited number of objects on a Remote form class (or a Window or Report class). (ST/HE/1766)

The following form or window object limits apply:

- **8191**
  object limit for a Remote form (or Window class), including objects on subforms, although in practice the limit is likely to be less due to platform limitations.
- **3000**
  The object limit for a Report class.

JavaScript Worker

The omnis_zip.js file has been moved into its own folder ‘omnis_zip’. (ST/EC/1752)

macOS System Font

The "Geneva" font entry in the font system classes has been replaced (in new libraries) with a new font labelled "Omnis macOS System" which is analogous to "Omnis Windows System" and represents the system font for the current platform.

External Components

On all platforms, external components are now loaded from the relevant sub-directory (xcomps, jscomps and logcomps folders) of both the Omnis data folder and the Omnis program / Application folder. (ST/*A/163)

If there is a component (.dll, .u_... or .so, depending on platform) with the same case-sensitive name in both relevant sub-directories of the data and program folders, the component in the data folder is loaded.

If you are upgrading to the latest version of Omnis Studio, and you have created your own external components for a previous version, then these components will need to be recompiled for the new version of Omnis Studio using the latest external component source files, which can be downloaded from the Omnis website.

FLDeditState

The **FLDeditState** structure has been modified, therefore all external components that use the WM_FLD_SETMENU message will need to be rebuilt when moving from Studio 10.2.x (and earlier) to Studio 11.

OLE2 Menu Options

The OLE2 menu options **Links, Object** and **Insert Object** have been removed from the **Edit** menu under Windows (these were used by OOLE2 which was removed in Omnis Studio 6.1).

DDE Menu Options

The DDE menu options **Paste Link** and **Remove DDE Link** have been removed from the Edit menu under Windows. These options can be reinstated by setting the new configuration item ‘includeDDEEditMenuItems’ in the ‘windows’ section of config.json to true (default value is false, i.e. the menu options are hidden). Omnis requires a restart after editing this item.

Web Client Form Cache

The $root.$clearcachedforms() method is no longer required and has therefore been removed from the design interface including the Property Manager (but the method will continue to work in existing apps for backwards compatibility). (ST/JS/2700)

The $root.$clearcachedforms() method was used to clear the cached forms for the Omnis Web Client plug-in which is no longer supported in Omnis Studio. The JavaScript Client does not cache forms in the same way, so this method is not required.

What's New in Omnis Studio 11 Revision 36251

The following enhancements have been added to Omnis Studio 11 Revision 36251. See the Readme.txt accompanying this release for information about faults fixed in this revision and for any last-minute release notes. See the Install.txt file to find out System Requirements for running the Development and Server versions of Omnis Studio.

Omnis Studio Now

Since the last general release of Omnis Studio, we have introduced Omnis Studio Now.

Omnis Studio Now is a new service that allows you to download the most recent revisions of Omnis Studio rather than having to wait for general releases. This allows you to receive new features and bug fixes earlier than being outside the program. Studio Now is available to developers on ODPP and RMA only.

The Omnis Studio Now service is available in a different version of Omnis Studio, using a different development serial number, which will unlock additional options within the Studio Browser. However, your customers will not require a new serial number (runtime or client license) to use the new features in the updated revisions of Omnis Studio available via Studio Now.

For more information about getting Omnis Studio Now, please contact your local sales team.

Studio Browser

A new **Studio Now** branch will appear in the Studio Browser tree that lists the latest Studio Now releases, including a list of enhancements and fixes for each release, plus links to the updated online docs for any enhancements. You can select which release to download, as well as the Platform (Windows, macOS, Linux), Architecture (64/32 bit) and the type of installer/file package (zip) you wish to download.

Documentation

The Omnis Studio online documentation will be updated for each Studio Now release and is available using the **Omnis Online Help** option in the **Help** menu inside Omnis Studio, or you can navigate to the online docs here. Any new features and enhancements, for all the latest revisions, are highlighted in yellow and marked with a revision number showing when the item was introduced.

JavaScript Components

JS Data Grid

Custom Picklists

A new client-executed method **$getpicklist()** has been added to the Data grid control to allow picklist columns to be specified per row. (Revision 35680, ST/JS/3350)

The $getpicklist() client method 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. The method has 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.

Resizing Columns

The **$resizecolumn** property has been added to the Data Grid control to specify which column is resized to fill the control width. (Revision 35849, ST/JS/2557)

The $resizecolumn property specifies 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.

For existing libraries, $resizecolumn is set to -1 (no column is resized) to maintain the behavior of previous versions. For new data grids, the default value is 0 meaning the last column is resized as appropriate.

JS Droplist

List line selection

The **$selectonopen** property has been added to the Droplist control to manage whether the first line is selected when a droplist is opened. (Revision 35731, ST/JS/3367)

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 from previous versions, whereby the first line was selected and evClick was 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.

List definition

The $listcolumn property of a Droplist can now be defined using a column name as well as the column number, as in previous versions. (Revision 36169, ST/JS/3449)

JS Combo Box

List definition

The $listcolumn property of a Combo Box can now be defined using a column name as well as the column number, as in previous versions. (Revision 36183, ST/JS/3462)

JS Rich Text Editor

Inserting text

The **$insertatcursor()** client-executed method has been added to the Rich Text Editor component to allow you to insert text into the field at the current insertion point. (Revision 35778, ST/JS/3381)

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.

Events

The standard **evBefore** and **evAfter** control events have been added to the Rich text editor control. Note that evAfter will only be triggered *if the contents have changed.* (Revision 36017, ST/JS/3419)

JS Native List

Themed SVG Icons

JS Native Lists now support themed SVGs. Therefore, the imagecol column in the data list for a Native List can contain a URL to a themed SVG (or PNG as in previous versions). (Revision 36075, ST/JS/3436)

Menu Icons

You can now add icons to the menus in a JS Native List. (Revision 35784, ST/JS/3379)

You can add icons to the rows in menus that are embedded in a Native List, by adding the following columns to the list defining the menu (specified in the $menulistname property):

- **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

JS Tile Grid

The following properties have been added to the Tile Grid control to provide tile shadow, grow on zoom, and image zoom capabilities. (Revision 35901 and 35876, ST/JS/3396 and others)

Property	Description	Revision	Fault
$tileshadow	adds a shadow to the tiles in the control	35876	ST/JS/3135
$tilegrowonhover	enables tiles to grow when the user's pointer hovers a tile	35901	ST/JS/3396
$tileimagezoom	zooms the image of a hovered tile, specified as the percentage by which the image expands	35901	ST/JS/3401
$horzpadding	specifies the left and right padding inside the tiles in the grid	35901	ST/JS/3402
$vertpadding	specifies the top and bottom padding inside the tiles in the grid	35901	ST/JS/3402

JS Toolbar

The **$toolbariconcolor** property has been added to the Toolbar control. (Revision 36016, ST/JS/3424)

The $toolbariconcolor property specifies the color of toolbar item icons. If set to kColorDefault, the icons will match the text color in the toolbar, otherwise the specified color is used.

JS Entry Field

evInput event

The **evInput** event has been added to the Entry Field to allow you to detect any change in the field content *without any key presses*, such as pasting in content. (Revision 36065, ST/JS/3426)

The evInput event is fired every time the value of the control changes as a direct result of a user action, such as, when the user has pressed a key, or cut or pasted text in the field. This is different from the evKeyPress event, which is triggered when the end user has pressed a key or keys.

Label Position

The **kJSLabelPosInside** constant has been added to the $labelposition property for Entry fields. When set to kJSLabelPosInside the label is positioned *inside the field border* when the cursor enters the field. (Revision 35990, ST/JS/3420)

Custom CSS Styles

The classes in $cssclassname have been added to the frame element of all JavaScript controls, with the "-frame" suffix. (Revision 35967, ST/JS/3416)

$cssclassname adds classes to the client element in all JavaScript controls, and now adds the same classes to the frame element with the "-frame" suffix.

Component Object Type

The **$objtype** property for JavaScript components (and window fields) is now displayed in Advanced mode only in the Property Manager (Revision 36160, ST/HE/1990)

For all JavaScript components $objtype is set to kComponent, but the type of object is displayed under the name at the top of the Property Manager, such as *Edit Control* for a single line edit field. Note you cannot change the type of an object.

JavaScript Remote Forms

Subform Sets

Trapping the close event

Dialog and palette style subforms in a subform set can now call the **$sfscanclose()** client method. (Revision 35677, ST/JS/3362)

Dialog and Palette style subforms, created inside a subform set using the "subformdialogshow" and "subformpaletteshow" client commands, 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.

Maximize behavior

The **kSFSflagAllowOutsideOfBounds** constant has been added to the subformset_add client command flags and the SFS maximize behavior has been improved. (Revision 35680, ST/JS/3360)

When using the subformset_add client command to create a subform set, the kSFSflagAllowOutsideOfBounds flag can be included to allow subforms to be positioned outside of their container boundaries, both by notation and the end user dragging them.

In addition, the maximize behavior has been changed to only fill the viewable area of the containing element, so there is no need for the end user to scroll to see all of the subform. The parent element, if scrollable, temporarily has its scroll ability disabled while a subform is in a maximized state.

Closing a subform with Esc

The **kSFSflagEscToClose** constant has been added to the subformset_add client command flags. (Revision 35747, ST/JS/2620)

When kSFSflagEscToClose is included in the flags for the subformset_add client command, the subforms in the set can be closed by pressing the Escape key.

Using a Promise with Client Commands

Various message type client commands now return a JavaScript promise when the methods are executed on the client. (Revision 35801, ST/JS/3380 & ST/JS/3384)

The **javamessage, yesnomessage,** and **noyesmessage** client commands (executed using the $clientcommand method), as well as the **$showmessage** method, now 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:});
```

Client Methods

Debugging Client Methods

You can now use the *Breakpoint* command in client-executed methods to allow you to debug them. (Revision 35949, ST/JS/3415)

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.

Server Method Calls

Multiple server method calls from client methods are now allowed and are queued; in previous versions, a client-side method could only call one server method at a time. (Revision 36084, ST/JS/3440)

If you try to call multiple server methods from a client method, Omnis still only allows one method to run at a time, but the others will now be queued. They will run in the order in which you call them. If your subsequent server method calls depend on previous server method calls, you should still take the "daisy-chain" approach of calling the next method from the previous method's ..._return callback method.

JS Theme

You can now specify a theme when opening a remote form in a browser. (Revision 35813, ST/JS/3386)

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
```

Vertical Text in PDF reports

When using the PDF Device to print reports to PDF in the JS Client, you can now print vertical text in a PDF report using the kEscAngle text escape and the kAngle90 or kAngle270 constants to rotate the text, for example, con(style(kEscAngle,kAngle270),iTextLine1). (Revision 36087, ST/EC/1572)

Note kEscAngle is not a new feature but support for creating vertical text in PDF reports has been fixed.

Window Components

Screen Report Fields

The **$print()** method has been enhanced allowing you to print from a Screen Report field to a PDF file. (Revision 35797, ST/RC/1425)

Two new parameters **bToPDF** and **cPDFPath** have been added to the $print() method to allow you to print a report in a screen report field to a PDF file. When bToPDF is kTrue and a path is specified in cPDFPath, a PDF file is created in the specified location. For example:

```
Do ScreenReportField.$print([bToPDF=kTrue,cPDFPath='<path>'])
```

If cPDFPath is empty, a prompt is shown allowing the end user to specify a path for the PDF file.

If the parameters are omitted or bToPDF is kFalse the method prints the report displayed in a screen report field to the current report destination, e.g. the Screen or Printer.

Combo Boxes and Droplists

Combo boxes and Droplists can now have border icons on the left side of the field, plus $contentpadding has been added to these components (Revision 35835, ST/WO/2781)

Combo boxes and Droplists now have the $bordericonstyle property which allows you to add an icon to the left side of the list. Note that border icons cannot be shown on right side due to the drop arrow icon in the list or combo box. The $studioide property must be set to kTrue to display border icons.

In addition, $contentpadding has been added to Combo boxes and Droplists to allow you to add padding around the content inside the controls.

Picture Controls

Picture controls now receive evClick and evDoubleClick events regardless of enter data mode. (Revision 35956, ST/WO/2798)

In previous versions, evClick and evDoubleClick were sent to the $event method in a Picture control *only in enter data mode*, while for some other controls (e.g. Lists, Edit fields) clicks are sent regardless of the enter data mode.

However, a click or double-click on a picture field will now receive events *regardless of enter data mode*. If you previously relied on this not happening, you can set $active to kFalse to prevent the events from triggering.

Entry Fields

Content Tips

Content tips in Entry Fields now allow styled text. (Revision 36125, ST/HE/1977)

When you enter the content tip in the Property Manager, a text editor allows you to select various styles including bold, italic, underline, and colors for the text. You can use the style() function to format the text when assigning a value to $contenttip, such as con(style(kEscColor,kRed),'Enter your last name').

HTML Controls

jOmnis.mDesign has been added to HTML controls which will be true in development mode. (Revision 36129, ST/EC/1840)

If you need to alter the behavior of your HTML control in some way in development mode, you can check the value of jOmnis.mDesign which will be true in development mode.

The Omnis Environment

Spell Checking

Words can now be added or removed from the custom dictionary when using spell checking for window fields in end user apps, and when using the Code Editor during development. (Revision 35711, ST/HE/1945)

When Spell checking is enabled for desktop apps and in the IDE ($showspellingerrors is true), the context menu for Edit fields has the **Learn Spelling** menu item when the selected word is shown as a spelling error. If this menu item is selected the word will be added to the end user's custom dictionary and will no longer show as a spelling error. Conversely, if a word has been added to the custom dictionary, the context menu will show the **Unlearn Spelling** menu item, which when selected will remove the word from the custom dictionary and the word will be shown as a spelling error.

These menu items are also available in the Code Editor context menu.

Property Manager

Setting Location and Size Properties

You can now change the **Location** or **Size** of an object in the Property Manager using the +, -, *, or / keys plus a number of pixels, for example, you can enter +20 in the $left property in the Property Manager to move the object 20 pixels to the right. (Revision 36049, ST/HE/1972)

The location and size properties appear in the top panel of the Property Manager and include the $left, $top, $width, and $height properties. This also works for a group of selected objects where the property value is the same for all objects in the group (if the value is different among the selected objects in the group, the property value is blank).

Key	Description	Example for $left
+n	Adds n pixels to property value(s)	+20 moves object(s) 20 pixels to the right
-n	Subtracts n pixels from property value(s)	-20 moves object(s) 20 pixels to the left
*n	Multiplies property value(s) by n	*2 doubles the value, moves object(s) to the right
/n	Divides property value(s) by n	/2 halves the value, moves object(s) to the left

Selecting Properties

The **Class Properties** and **Field Properties** options have been added to the Remote form class (and Window class) context menu to allow you to select the properties of the current Class or Field as required. This replaces the single Properties option available in previous versions. (Revision 36209, ST/HE/1987)

Changing Boolean Properties

You can double-click on a Boolean (kTrue/kFalse) property value in the Property Manager to toggle its value (as well as clicking the switch). No other properties can now be changed by double-clicking, as in previous versions; in this case, double-clicking will now select the current value. (Revision 36173, ST/DB/1473)

Save Window Setup and Omnis Preferences

The position information saved for various screens in the IDE has been moved from omnis.cfg to a new file called positions.cfg: note you cannot edit positions.cfg or omnis.cfg. (Revision 36199, ST/IN/294)

The positions.cfg configuration file holds the position information saved for various screens in the IDE using the Save Window Setup option. This ensures that the IDE screens are returned to their saved size and position when you reopen Omnis. The information includes window positions and sizes, split bar positions, etc, for each screen layout.

The omnis.cfg configuration file holds all the other settings saved with Save Window Setup, e.g. Show tree for the method editor window. There is a single value stored in omnis.cfg for all screen layouts.

In addition, some properties in the Omnis Preferences ($root.$prefs) have been replicated in a new “prefs” group in the Omnis configuration file (config.json), which means you can now set their values in either the Property Manager or the Configuration Editor. The new “prefs” group has the following items:

```
"prefs": {
        "allowEditIfNotCheckedOut": false,
        "disableReportCopy": false,
        "disableReportWorkingMessage": false,
        "disableSystemIdleSleep": false,
        "disableSystemIdleSleepReason": "Omnis Studio is busy",
        "exportBOM": true,
        "exportEncoding": "kUniTypeUTF8",
        "helpBarOn": false,
        "importEncoding": "kUniTypeUTF8",
        "listSearchTimeout": 40,
        "mapDMLtoDAM": "Disabled",
        "maxCachedClasses": 1024,
        "mouseWheelLines": 3,
        "oldListHiliting": false,
        "oldListSearching": false,
        "promptForReorg": true,
        "reportToolbarPagePreview":
  "kRBpageList+kRBprint+kRBprintPage+kRBsave+kRBsavePDF+kRBsearch+kRBzoom",
        "showToolbarTips": true,
        "showWindowTips": true,
        "useCms": true,
        "webBrowser": ""
    }
```

You can edit config.json using the **Edit configuration** option available in the Options menu in the bottom-left corner of the Studio Browser.

Unicode

Import Encoding

The default value of the **$importencoding** Omnis preference has been changed to kUniTypeUTF8, to match the value of $exportencoding; it was set to kUniTypeNativeCharacters in previous versions. (Revision 36227, ST/NT/814)

Note that “importencoding” and “exportencoding” can also be set in the new “prefs” group in the Omnis configuration file (config.json).

Libraries and Classes

Export JSON Options

The 'fullexportimport' option has been added to $exportimportjsonoptions to control what is exported (or imported) when a library is exported to JSON; the same info is stored the 'exportimportjsonoptions' group in config.json. (Revision 35844, ST/IE/232)

The 'fullexportimport' option in $exportimportjsonoptions defaults to true, which means all library information is included in the JSON export (maintaining behavior in previous revisions).

If the option is set to false, Omnis does not export certain information which is not required to represent the library, including 'internalversion', 'omnisbuild' and 'moddate'. A library exported with fullexportimport set to false can only be imported if fullexportimport is set to false.

List Programming

Defining Lists

The library preference $defineresolvesfieldrefs has been added to manage how field references behave when used for defining lists. (Revision 35747, ST/NT/811)

When **$defineresolvesfieldrefs** is set to kTrue (default is kFalse), if a field used to define a list is a field reference, Omnis resolves the field reference and defines/redefines the list using the resolved field.

In previous revisions, the *Define list* command and the $define method behaved inconsistently when the variables used to define the list in a called method are field reference parameters.

Binary Data in Lists

The limit of 100MB for Binary data in a list variable has been removed. (Revision 35692, ST/PF/1406)

Server-Specific Programming

PostgreSQL

Notification Channels

The $listenername and $cannotify properties have been added to the PostgreSQL DAM. (Revision 35971, ST/*P/124 and Revision 35977, ST/*A/169)

Similar to the $programname session property, you can assign or change the name for the listener session using the new **$listenername** property. This name will subsequently appear in the pg_stat_activity system table.

To prevent incoming notifications from interrupting the currently executing method, you can use the new **$cannotify** property. Setting this to kFalse, disables the $notify() method and causes incoming notifications to be queued. Set $cannotify to kTrue to receive any queued notifications.

Debugging Methods

Method Stack Limit

A new item **stackLimit** has been added to the “default” section of the Omnis configuration file (config.json) that allows you to set the limit on the number of methods allowed on the method stack. (Revision 35840, ST/PF/1409)

You can control the number of methods on the Omnis method stack by setting the stackLimit item in the “default” section of the Omnis configuration file (config.json); the default value is 30 which is adequate for most applications. Omnis fetches the value of stackLimit on startup, therefore when a library is opened, the stack limit is already in effect.

The method stack size has been increased on Windows to 8MB to accommodate more items on the stack, bringing it into line with macOS and Linux. (Revision 35840, ST/PF/1412)

Note the function sys(290) returns the number of methods on the method stack, while sys(192) returns a list of methods on the method stack.

OW3 Worker Objects

Hash Worker Object

The **$initverifysignature()** method has been added to the Hash Worker Object to allow you to verify the RSA signature from $initsignature (Revision 36002, ST/EC/1790)

The $initverifysignature() method allows you to verify a signature from $initsignature.

```
$initverifysignature(vData,iHashType,vPublicKeyPEM,vSignature)
```

The method takes vData the original data, iHashType the original hash type, vPublicKeyPEM the public key in PEM format and vSignature the signature from $initsignature to verify the signature.

When returning to $completed, the row's errorCode will be 0 if the signature has been verified successfully (that is, the data has not been tampered with and it matches the signature), otherwise it will have a mbedtls or an Omnis error code if something has gone wrong (e.g. if the signature doesn't match, it should return -17280 with error info of ''RSA - The PKCS#1 verification failed'').

OAUTH2 Worker Object

The **$tokentype** property has been added to the OAUTH2 Worker Object to provide a fallback when endpoints do not return a valid token_type. (Revision 36024, ST/EC/1831)

The $tokentype property is a string with the value 'Bearer' which is used as a fallback token type if the OAUTH2 worker object does not return token_type in its response. This should only be used when you’re sure it's needed and as a workaround for a bad or missing OAUTH2 endpoint.

Commands

Importing Data

The new item 'LFonlyLineTermination' has been added to the Omnis Configuration file (config.json). (Revision 35879, ST/IE/234)

The 'LFonlyLineTermination' item in the 'default' section of config.json allows you to control how carriage returns and line feeds are handled when importing data from a file. If true, when Omnis imports a tab- or comma-separated file and the file has no Carriage Return (CR) line separators, Omnis will then check for Line Feed (LF) line separators and use these to break record rows.

Functions

FileOps Workers

Three new Worker Objects have been added to the FileOps external component to allow you to Copy, Move or Delete files asynchronously if required – in essence they work the same as their equivalent FileOps functions that can operate on a separate thread. (Revision 35801, ST/EC/1770)

The FileOps Worker Objects, **FileOpsCopyWorker, FileOpsMoveWorker,** and **FileOpsDeleteWorker** allow you to copy, move or delete files asynchronously, which may be useful when operating on a large number of files.

To use the FileOps workers, you need to create an Object variable and set its subtype to one of the FileOps worker objects via the Select Object dialog. Alternatively, you can create an Object class and set its superclass to one of the FileOps worker objects, then create an Object variable or Object reference variable and set its subtype to the object class name. Having created the variable you can call its methods using OBJECTVAR.$methodname.

The FileOps workers work exactly like the other workers, including the ability to run the $init, $start or $run methods. They have the property $progressinterval which sets an interval in seconds, at which progress notifications are sent to the $progress method. $progressinterval defaults to the minimum value of 1 second.

For the $init method, the parameters are the same as used in the static methods of the FileOps object to *copyfile, movefile* or *deletefile*.

When $completed is called, a row is returned with the errorCode and errorText of the action; errorCode will be 0 if there is no error, otherwise a FileOps Worker error is returned, plus the error description in the errorText column.

When the $progress method is called (only if $callprogress is true), a row is returned with a column name 'progress' containing an estimated number between 0 and 100 indicating the percentage progress of the file operation. Note: the Delete worker never calls $progress, therefore progress notifications are not sent from the delete worker.

Note that if you try to cancel a FileOps worker while it is running, the main thread will be blocked until the worker finishes its job.

FileOps.$spaceinfo()

The **FileOps.$spaceinfo()** static method has been added to return disk space information about the specified file system. (Revision 36134, ST/FU/857)

Syntax

FileOps.$spaceinfo(*cPath,&iSize,&iFree,&iAvailable*)

Description

Returns disk space information about the file system specified in *cPath*. *iSize, iFree* and *iAvailable* are 64-bit integers, in units of bytes.

Returns Boolean true if size information was returned.

split()

The **kComma** constant has been added and can now be used in the *split()* function to define the delimiters instead of using ',' (the comma character). (Revision 35919, ST/FU/877)

The kComma constant has been added to the Special Characters group of constants that can be used to represent a comma (ASCII character 44).

The syntax for the split() function has changed replacing the parameter delimiters=',' to delimiters=kComma. The full syntax is now:

```
split(string[,delimiters=kComma,stripWhite=kFalse])
```

bitset() and bittest()

The *bitset()* and *bittest()* functions can now be executed on the client. (Revision 36036, ST/JS/3429)

When executing on the client, the binary argument must be a 32-bit integer variable.

As *bitset()* modifies the 'binary' parameter in place, this must be given a VARIABLE directly, not a literal or the result of a calculation. If you try to do that, you will get an error describing this.

In addition, the *lastBitNumber* parameter in the *bitset()* and *bittest()* functions is now optional. If *lastBitNumber* is omitted, the value in the *firstBitNumber* parameter is used allowing you to set/test a single bit. (Revision 35930, ST/FU/880)

bitclear()

The *bitclear()* function can now be executed on the client. (Revision 36036, ST/JS/3429)

When executing on the client, the binary argument must be a 32-bit integer variable.

As *bitclear()* modifies the 'binary' parameter in place, this must be given a VARIABLE directly, not a literal or the result of a calculation. If you try to do that, you will get an error describing this.

iso8601toomnis() and omnistoiso8601()

The *iso8601toomnis()* and *omnistoiso8601()* functions can now be executed on the client. (Revision 35925, ST/JS/3404)

hexcolor() and hsla()

The *hexcolor()* and *hsla()* functions can now be executed on the client. (Revision 35914, ST/JS/3271)

dpart()

The *dpart()* function can now be executed on the client. (Revision 35991, ST/JS/3418)

truergb()

The *truergb()* function can now be executed on the client. (Revision 36007, ST/JS/3423)

Deploying your Web & Mobile Apps

Headless Server Admin Tool

The Upload button has been removed from the Admin Tool (osadmin) for the Headless Omnis Server. You can configure deployment files using a Dockerfile in Docker; see the Tech notes for more information. (Revision 35891, ST/AD/243)

Omnis Studio External Components

OmnisObject and OmnisObjectContainer

OmnisObject and OmnisObjectContainer have been moved into extcomp.he. (Revision 35680, ST/EC/1812)

OmnisObject is a base class that can be used for representing an external object that is being implemented. OmnisObjectContainer is a container for OmnisObject objects which could then be cast back to the external object class when needed.

These base classes provide a starting point and a container for multiple external component classes – you could offer more object classes from one external component project (e.g. workers and non-visual objects) and keep your classes in a OmnisObjectContainer.

Furthermore, these track the number of references internally and delete themselves when references reach 0, in line with other Omnis external components.

Version Support

Two new functions **ECOmeetsStudioVersion** and **ECOisStudioNow** have been added to the External Components to check the Omnis version. (Revision 36168, ST/DC/989)

qbool ECOmeetsStudioVersion(qshort pMajor, qshort pMinor) returns qtrue if the current Omnis major and minor versions meet or exceed the passed pMajor and pMinor versions, else returns qfalse.

qbool ECOisStudioNow() returns qtrue if Omnis is Studio Now, else returns qfalse.

ToolTips

Two new functions **ECOsetToolTipText** and **ECOgetToolTipText** have been added to the External Components interface to support longer tooltips (>255 characters) in your externals. (Revision 36112, ST/EC/1839)

If you wish to use more than 255 characters in tooltips in your Omnis externals, you can use ECOsetToolTipText(TIPinfo* pTooltip, EXTfldval& pText). Pass in the TIPinfo* as first parameter and a EXTfldval with the text you wish the tooltip to use.

Note ECOsetToolTipText will change the TIPinfo->mTipType to TIPinfo::eFldval and will clear mToolTipText. Furthermore, ECOsetToolTipText will set the TIPinfo->mHasTip member to qtrue, therefore removing the need to do so after using ECOsetToolTipText.

You can use ECOgetToolTipText(TIPinfo* pTooltip, EXTfldval& pOut) to get a previously set tooltip. You need to pass a TIPinfo* as the first parameter and an emtpy EXTfldval as the second parameter to receive your tooltip text. Note ECOgetToolTipText is designed specifically for tooltips set with ECOsetToolTipText, where TIPinfo->mTipType == TIPinfo::eFldval (because the fldval is held in the core, therefore it would not be accessible after you've set it, if you wanted to check it again).

Version Control

VCS API

The VCS API now allows you to check out specific revisions of a class (Revision 36146, ST/VC/820)

A new method **$x_listClassRevisions** has been added to the VCS API to list class revisions.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_listClassRevisions',rClassRef,lClassRevList,cToken,cErrors)
   Returns bStatus
```

refClassRef is a reference to a class in your local library. If the call is successful, a list of revisions will be returned in **lClassRevList.** The first column of this list will contain the revision number.

If you wish to copy out a revision, use a revision number from lClassRevList in a new parameter **revID** added to $x_checkOut to check out the revision. For example:

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_checkOut',refClassRef,refLibRef,cToken,bCheckOrCopy,
   revID,cErrors) Returns bStatus
```

What's New in Omnis Studio 11 Revision 35659

The following enhancements have been added to Omnis Studio 11 Revision 35659. See the Readme.txt accompanying this release for information about faults fixed in this revision and for any last-minute release notes. See the Install.txt file to find out System Requirements for running the Development and Server versions of Omnis Studio.

Libraries and Classes

Class Locking and Library Conversion

In order to enhance the integrity and security of deployed Omnis Studio libraries, the mechanism used to lock classes in a private library has changed in Omnis Studio Revision 35659.

Consequently, all libraries opened in Omnis Studio 11 revision 35659 or later **WILL REQUIRE CONVERSION, INCLUDING LIBRARIES CREATED WITH ALL PRIOR REVISIONS OF OMNIS STUDIO 11** (as well as Studio 10 or earlier libraries). THE LIBRARY CONVERSION PROCESS IS IRREVERSIBLE.

THEREFORE, AND IN ALL CASES, YOU SHOULD MAKE A SECURE BACKUP of all existing Omnis Studio 11 libraries BEFORE OPENING THEM in Omnis Studio 11 Revision 35659 or later.

JavaScript Components

JS Camera

All JS Camera events now have an additional pError param which reports any possible errors. (ST/JS/3339)

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

JS File

Two new properties $choosefilesbuttontextpos and $choosefilesiconid have been added to the JS File component to allow more control over how the UI is displayed in Upload mode of the control. (ST/JS/3322)

The **$choosefilesbuttontextpos** property allows you to position the text label in the Upload UI for the File component. 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 for the File component; 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.

Native List

A new property $reordermode has been added to the JS Native List component to allow end users to reorder rows within the list. In addition, there is a new property $reorderbetweengroups to control whether rows can be reordered between groups. (ST/JS/3327 and ST/JS/3341)

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 Native List component has a new event **evReorder** to report 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

JavaScript Remote Forms

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 now taken into consideration. (ST/*L/055)

Date/Time data fetched from a SQLite database were previously sent to the client as full datetimes. However, now 'Short Date...' data subtypes will return a date only (no time component), and 'Time' data subtypes will return a time only (no date component).

Window Components

Complex Grid

A new event evColumnDividerMoved has been added to the Complex Grid control to report when a grid divider has been dragged. (ST/GR/444)

The **evColumnDividerMoved** event is sent to a Complex Grid when a column divider has been dragged. The event has two parameters:

- **pDivider**
  the divider column number (same ID as in $dividers)
- **pDividerMoveBy**
  the number of pixels the divider was moved by; note this can be negative

You can discard the event to stop the grid divider being moved.

Toolbars

The $splitbuttonbars property has been added to Toolbar classes to allow you to split radio button groups into individual button items – in previous versions, contiguous radio buttons were displayed in a compact group and only the text label for the first button was displayed. (ST/TB/340)

On macOS, if **$splitbuttonbars** is set to kTrue, a group of radio buttons (of kToolRadioButton type) will be displayed as separate buttons with their respective labels; the default is kFalse (to maintain backwards compatibility) where radio buttons are displayed in a compact group. The property only applies to toolbars on macOS, including main toolbars, floating toolbars and for unified window toolbars where kTBOptionmacOSOmnisTopToolbar is true.

oBrowser

Chromium Safe Storage Prompt (macOS only)

On macOS, when first running an updated version of Omnis Studio, where a previous version was installed, the user will be presented with a prompt when oBrowser is loaded: "*Omnis wants to use your confidential information stored in "Chromium Safe Storage" in your keychain – To allow this, enter the "login" keychain password.*"

The Chromium Safe Storage keychain item is used to grant access to the shared Chromium encrypted data store for secure storage of cookies. It is recommended that access is granted to allow cookies to be encrypted.

To address this, support for the "use-mock-keychain" CEF switch has been added to the "obrowser" section of the Configuration file (config.json). (ST/IN/286).

The prompt can be disabled by adding the use-mock-keychain CEF switch to the Omnis configuration. To do this, add the following entry to the 'obrowser' section of config.json:

```
"obrowser": {
   "cefSwitches": [
      "use-mock-keychain"
   ]
}
```

It is important to note that if this is disabled cookies will NOT be secure.

Omnis Environment

Omnis Configuration

Keystroke Receiving Prompt (macOS only)

On macOS, when first running a new version of Omnis Studio, the end user will be presented with a prompt: "*Omnis Studio 11 would like to receive keystrokes from any application – Grant access to this application in Privacy & Security settings, located in System Settings.*"

This is required to provide full keyboard support to Omnis Studio for monitoring events from the macOS Dock and Mission Control. Access can be granted when prompted for Keystroke Receiving, or you can ensure there is an entry granting access in the Input Monitoring section of the Privacy system setting.

To address this, a new item called **monitorDockKeyEventsInfoPrompt** has been added to the "macOS" section of the Configuration file (config.json). (ST/IN/286).

To show a one-time only prompt in Omnis Studio, *prior to the system prompt*, set the **monitorDockKeyEventsInfoPrompt** config entry to true (the default is false). The message can be customized by changing the entry for CORE_RES_19003 in /Contents/Resources/[LOCALE].lproj/Localizable.strings

Keystroke receiving and the access prompt can be disabled by changing the "monitorDockKeyEvents" to false in config.json.

When Omnis Studio does not have full keyboard support, the order of windows displayed may not be correct after using Mission Control with the keyboard.

Web and Email Communications

OAUTH2 Worker Object

The $oauth2state property has been added to the OAUTH2 Worker Object to allow you to append custom content to the state query string parameter. (ST/EC/1803)

The **$oauth2state** property can contain custom content to be appended to the 32-character UUID in the state query string parameter of the request, allowing you to identify requests sent from multiple instances of Omnis.

If you are handling this on a reverse proxy, you will have to URL-decode and look for your value after the first 32 characters, but it is important when proxying off the request to keep the UUID in the state, otherwise Omnis will not be able to match the callback to the initiated request.

SMTP Worker Object

A new property $allowpathinuri has been added to the SMTP Worker Object to allow a path in the URI. (ST/EC/1808)

When the **$allowpathinuri** property is true (the default is false), the SMTP Worker will accept paths in the URI used in the $init method, e.g. smtp://my.smtp.server:587/my.helo.address.

Functions

OIMAGE.$makeqrcode

A new parameter wCenterImg has been added to the OIMAGE.$makeqrcode function to allow you to add an icon image to the center of the generated QR code. (ST/FU/863)

The **wCenterImg** parameter controls how an icon can be overlaid after a QR code has been created; this only works for kOIMAGEfmtPNG output QR images (not for kOIMAGEfmtSVG). The new parameter is a row variable with the following parameters:

```
row(cIconID,[ixSize=15,ixBorder=2,ixForeColor=kColorBlack,ixBackColor=kColorWh
   ite])
```

By default, the row requires a single parameter, **cIconID,** which is the icon to be overlaid in the center of the QR code (this can be an SVG or PNG icon image). The second parameter **ixSize** controls the size of the icon, which is a percentage of the output QR image size; this defaults to 15% with a maximum value of 30% (a value over 30% may result in an error).

A border can be added to the overlaid icon using a third parameter **ixBorder** which defaults to 2 pixels. In addition, you can control the center icon area color, and icon color using parameters **ixForeColor** and **ixBackColor,** which default to black and white (to match the resulting QR image which is black and white).

For example, to add the account_box icon to the center of a QR code using the default size and border, use the following command:

```
Do OIMAGE.$makeqrcode(
   lText,iPicture,kOIMAGEfmtPNG,256,
   iErrorLevel,4,errText,row('account_box'))
```

To set the size of the overlaid icon to 25% of the QR image size, add a 5 pixel border, and to set the colors of the overlaid icon and border color, use the following command:

```
Do OIMAGE.$makeqrcode(
   lText,iPicture,kOIMAGEfmtPNG,256,
   iErrorLevel,4,errText,row('account_box',25,5,kWhite,kRed))
```

Note the account_box icon is a themed icon so you can set the icon color (but remember the output QR image must be a PNG, i.e. kOIMAGEfmtPNG).

rnd()

A new parameter asnumber has been added to the rnd() function to force Omnis to return a Number, rather than a Character value, which is the default in previous revisions. (ST/FU/864)

The syntax is now:

rnd(*number*,*dp*[,*asnumber*=kFalse])

When set to kTrue, the parameter **asnumber** specifies that the function should return a Number, otherwise a Character value is returned when the parameter is omitted or set to kFalse. Note that this may lose some precision if the data type cannot represent the full floating-point value.

mouseover()

You can now return a reference to the field under the mouse in a Complex grid using a new parameter *tablefield* with the mouseover() function. (ST/GR/446)

The syntax is now:

mouseover(constant[,tablefield])

A new parameter **tablefield** has been added to the mouseover() function, so when used with kMItemref and tablefield is kTrue, the function will return a reference to the field under the mouse in a Complex grid.

FileOpsObj

The $filepath property has been added to the FileOps object to allow you to obtain a path to the file associated with the object. (ST/EC/1805)

After a file has been opened, e.g. after using $createtmpfile(), the **$filepath** property of the FileOps object instance will contain the path to the file. This is an alternative or shortcut to returning the file name in the list provided by the $getfileinfo() function. If there is no open file or the file has been closed, $filepath is empty.

Online Documentation

Latest Revisions

All the enhancements in Revision 35659 of Omnis Studio (and the previous revision 35439) are highlighted in yellow in the online documentation to show you what has been added.

What's New in Omnis Studio 11 Revision 35439

The following enhancements have been added to Omnis Studio 11 Revision 35439. See the Readme.txt accompanying this release for information about faults fixed in this revision and for any last-minute release notes.

JavaScript Components

JS Camera Control

The **$showui** property has been added to the Camera control and allows you to show the appropriate UI for using the Camera, Barcode scanner, and to switch between the front and back camera. (ST/JS/3313)

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, to maintain compatibility with the previous version), **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.

In addition, 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.

JS Data Grid

The **pIsNewRow** parameter has been added to the evCellValueChanged event for JS Data Grid component. If pIsNewRow is true, the cell belongs to a new row. (ST/JS/3292)

JS Native List

The **$dateformat, $dateformatcustom** and **$numberformat** properties have been added to the JS Native List. (ST/JS/3268)

These properties work in the same way as they do in other components, such as Edit fields, Lists, etc, and allow you to set the date and number format for the Native list.

List Pager

You can now 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 control. The timer for the long press is hard coded to 700ms. (ST/JS/1379)

JavaScript Remote Forms

PDF Printing

The **kDevOmnisSubsetNone** constant has been added to the $setpdfsubset() method in the Omnis PDF Device to unset the PDF subset. (ST/EC/1786)

You can set the PDF/A subset type for a PDF file using the $setpdfsubset() method, which is used to create archival versions of documents. If you wish to unset the PDF subset, you can now use the kDevOmnisSubsetNone constant with the $setpdfsubset() method.

Push Notifications

The Push Notifications library has been updated to support Firebase HTTP API v1. (ST/PC/591)

Firebase has deprecated their original Push Notification API, and this will stop working from April 2024. Consequently, the Push Notifications library available for Omnis Studio 11 has been updated to support Firebase HTTP API v1. Further information on how to transition to the new API can be found in the Push Notifications docs, available with the JS Wrapper download:
https://omnis.net/developers/resources/download/jswrapper.jsp

Window Components

Headed List

The $autosizecolumn() method has been added to the Headed list window control. The $autosizecolumn(iColumn) method resizes the specified column in the headed list, based on the maximum width of the data in the column. (ST/WO/2720)

Popup List

You can now open a Popup list with either the **Space** or **Return** key. In addition, on macOS, you can close a popup list with the Space key. (ST/WO/2766)

When a popup list is closed, you can navigate through its values using the **Up** and **Down** arrow keys to select a value. In this case, an evClick event will be sent as you navigate up and down (the same as for Combo boxes).

oBrowser

The **$htmlcontrolsusehttp** property has been added to the OBrowser control, to force Omnis to use the built-in web server to serve html controls over HTTP. (ST/EC/1795)

The $htmlcontrolsusehttp property controls whether html controls are served using a file:// URL (when set to kFalse), or using the built-in HTTP server (kTrue). File URLs are not deemed secure, so some web APIs may not be available. Therefore, it is recommended that you set $htmlcontrolsusehttp to true. When true, html controls are loaded from the 'html/htmlcontrols' folder in Omnis.

The config options "htmlcontrolsFolder" and "defaultHtmlcontrolsFolderInDataFolder" only apply when $htmlcontrolsusehttp is false.

Multibutton Control

When specifying the icons to be used with the **Multibutton** control using the $iconstr property, you can now include the icon size. (ST/IF/357)

In general, you should use SVG images for button icons, including the Multibutton control and all other controls, to achieve good scaling of icon images. This enhancement applies when using PNG images and allows you to choose the best size PNG icons for your Multibutton control.

The $iconstr property is a comma separated list of icon ids that are displayed when the Multibutton control is opened, that also specifies the number of button options in the control. The icon id for PNG icons can now use the form <id>x<size> where size is one of the available standard icon image sizes, that is, 16, 32, or 48. For example, 2033x48 can be used to specify the bitmap icon with ID 2033 and its 48x48 version. If no size is specified, then the default icon size is used.

Libraries and Classes

Class Data and Method Text Notation

Two new library preferences, **$disableclassdatanotation** and **$disablemethodtextnotation** have been added to stop other Omnis libraries accessing class data and/or method code within the library. They can be set using the Property Manager or via the VCS when building a library. They are defined as:

Property	Description
$disableclassdatanotation	If true, $classdata for all classes in the library will not be accessible. **Setting this property is an irreversible operation.**
$disablemethodtextnotation	If true, $methodtext and $methodlines will not be accessible. **Setting this property is an irreversible operation.**

When $disableclassdatanotation is kTrue for a library, you will no longer be able to read or write $classdata from any Omnis class using Omnis code. In addition, JSON export of the library is disabled as the class data is disabled. IMPORTANT: future access to this library in the Omnis VCS will no longer be possible.

When $disablemethodtextnotation is kTrue for a library, you will no longer be able to read or write $methodtext or $methodlines from Omnis code or via the Property Manager. In addition, method text will not be exported during a JSON export of the library.

Version Control

Building Projects

When building a library, you can now block access to the class data and/or method text in the library by setting the new options **'Disable Class Data Notation'** and **'Disable Method Text Notation'** (these options set the new $disableclassdatanotation and $disablemethodtextnotation library preferences: see the previous section for more information about these new properties).

IMPORTANT: Setting either of these properties is irreversible in the built library and future access to this library in the Omnis VCS will no longer be possible.

VCS API

The **$x_listProjectClasses** method has been added to the VCS API to return a list of classes in the specified project. (ST/VC/803)

The $x_listProjectClasses method returns a list of classes in the specified project, and has the following syntax:

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_listProjectClasses',cLIBNAME,lClassList,cToken,cErrors)
   Returns bStatus
```

cLibName is the name of the project in the VCS from which you want to return the class list.

If the call is successful, **lClassList** will return a list containing the following information:

className, classType, classVersion, classRevision, status (0 - checked in, 1 - checked out), checkedOutDate, checkedOutBy, checkOutNotes, checkedInDate, checkedInBy, checkedInNotes.

oProcess

The **oProcess** external no longer attempts to execute the special callbacks $stdout, $stderr or $started when launched via $run. These are now used only when launched via $start. (ST/EC/1784)

When using oProcess with $run, you can use either $readlines or $readtail to get the stdout or stderr results in the $completed callback.

Web and Email Communication

Python Worker

$init method

The **$init** method of the OW3 **Python worker** now accepts a second optional parameter, cExecutable, which is a character variable containing the path to the Python executable to use, overriding the default location of the Python executable. Without the new parameter the Python worker looks in /usr/bin/python3 for the Python executable. (ST/EC/1794)

Example app

A **Python Worker** example app has been added to the **Samples** section in the **Hub** in the Studio Browser.

LDAP Worker

An example app called **LDAP Client Worker Object** has been added to the **Samples** section in the **Hub** in the Studio Browser.

What's New in Omnis Studio 11

The IDE in Omnis Studio 11 has been greatly enhanced and updated, including the Studio Browser, Property Manager, Component Store, Catalog, Remote Form editor, as well as many of the other tools and class editors. Taken together, we believe these new enhancements make it quicker and easier to create web and mobile applications using Omnis Studio 11.

The following enhancements have been added to Omnis Studio 11.

- **Enhancements in the IDE**
  the layout and overall appearance of the **Studio Browser** has been dramatically improved making it easier to use and to navigate; the layout of the **Property Manager** has also been improved, with the common properties for an object appearing in a new top panel; the **Component Store** has been completely redesigned with the components grouped in a vertical panel; an extra column has been added to the **Catalog** window to show the values in the currently selected tab; there is a new editor for the **Omnis Configuration** file; Omnis now checks spelling in design mode, and in end user desktop forms (not JS remote forms); Omnis now supports multiple Undo and Redo operations; and IDE themes and colors have been adapted to support Dark Mode in Windows and macOS.
- **JavaScript Components**
  there are several new JavaScript components including: **Chart, Gauge, Camera, Floating Action Button, Tile Grid, Scroll Box** and **Color Picker,** plus many of the JS controls have been enhanced including Data grids, Edit fields, buttons and the File control; support for **Side Panels** has been added to some JS controls; you can add **Icon Badges** to some JS controls to show notifications; plus there are many new sample apps in the Hub in the Studio Browser showing the new JS controls and other enhancements (check the New filter option).
- **JavaScript Remote Forms**
  the Design bar on the Remote Form editor now has a Methods button and a Test button; a new option allows you to select which web browser to use to test a remote form; you can now open a remote form as a **Subform Palette,** which will pop out next to a specified control; the **$eventclient** method has been added to support client-executed methods for specific named events; and Remote Forms can now have only one layout breakpoint if required.
- **Libraries and Classes**
  Omnis now opens any libraries that were open at shutdown when it next starts up; the File menu in the Runtime and Server versions of Omnis has a new option to Close all Libraries; there is a new task message sent after a library or libraries have been opened or closed; extra validation when naming an Omnis class has been added to prevent certain characters from being used; and there is a new method to expose the methods in one library to be used in another library.
- **Method Editor**
  you can now add conditions or a hit count to Breakpoints, plus Breakpoints are restored the next time you open a library; a search panel has been added to the List variable window to allow you to search in the contents of a large list; and a new option Find Possible Calls... allows you to locate all possible calls to a method.
- **System Notifications**
  Omnis can now send notifications to the operating system on the end user's

computer, on both Windows 10/11 and macOS; notifications will pop up on the end user's screen and are added to the Notification Center for the current OS.

- **Power Management Notifications**
  Omnis Studio can now receive sleep and wake notifications from the operating system to indicate power management changes, so requests from the system to go into idle sleep can be denied on macOS or disabled.
- **Windows Components**
  you can add icons to the left and right border of entry fields to provide a visual hint to the end user; Content tips now animate above an entry field as the field gets the focus; you can add a set of buttons to each row in a list or headed list; you can now define Tab Groups in a vertical tab strip; and you can use Themed SVG icons for window controls including icons from the material iconset.
- **Omnis VCS API**
  some functions in the Omnis VCS have been exposed via API calls, to allow you to interact with the VCS or its contents programmatically.
- **Report Programming**
  you can now specify a Zoom factor for reports sent to Page preview; the Report list component calculation properties are now tokenized so that they work with the current function parameter separator; and PDF Printing now uses Node.js to print reports, rather than Python
- **User Constants**
  a new User Constants class allows you to define your own constants for use in your methods and expressions. A user constant is a named value, where the value cannot be changed during execution.
- **OW3 Workers**
  New LDAP and Python Workers have been added to the OW3 group of worker objects; plus support for HTTP/2 has been added for the OW3 Workers which is more secure as it uses binary protocols instead of plaintext, and is generally faster and more efficient for web communication.
- **Functions: OIMAGE, TOTP**
  there is a new external called OIMAGE that contains a set of static functions that allow you to resize JPG and PNG image files or perform various transformations on the images including rotation and flipping; plus two new functions have been added to the OW3 package to generate and validate Time-based One-Time Passwords (TOTPs); there are new sample apps in the Hub for OIMAGE and TOTPs.
- **Deployment Tool**
  A number of methods have been exposed in the Deployment Tool API to allow you manage builds in your own code, rather than via the Deployment Tool UI (the Deployment Tool is for desktop apps only)

The Omnis Environment

Enhancements in the IDE

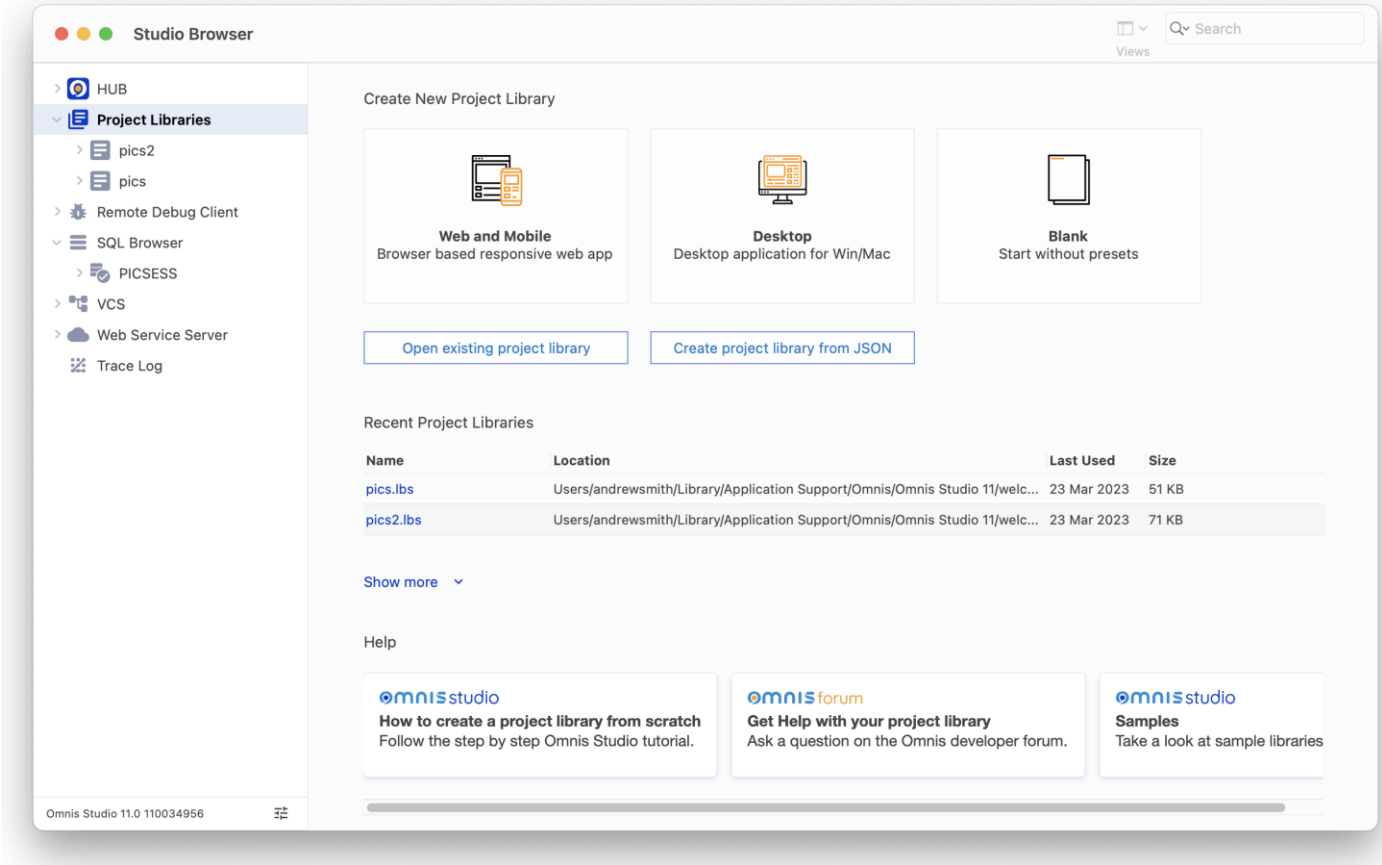
The IDE in Studio 11 has been greatly enhanced and updated, including the Studio Browser, Catalog, Property Manager, Remote Form editor, Component Store, as well as many of the other tools and class editors.

In general, lists and text labels in the IDE have more padding, colors and text have been softened, and all icons in the IDE have been redesigned.

Taken together, we believe these enhancements make it quicker and easier to create web and mobile applications using Omnis Studio 11.

Studio Browser

When you start Omnis Studio 11, you'll immediately see the enhancements we've made to the **Studio Browser.** The mechanics of creating and managing libraries are largely the same, but we have dramatically improved the layout and overall appearance of the Studio Browser to make it easier to use and to navigate.



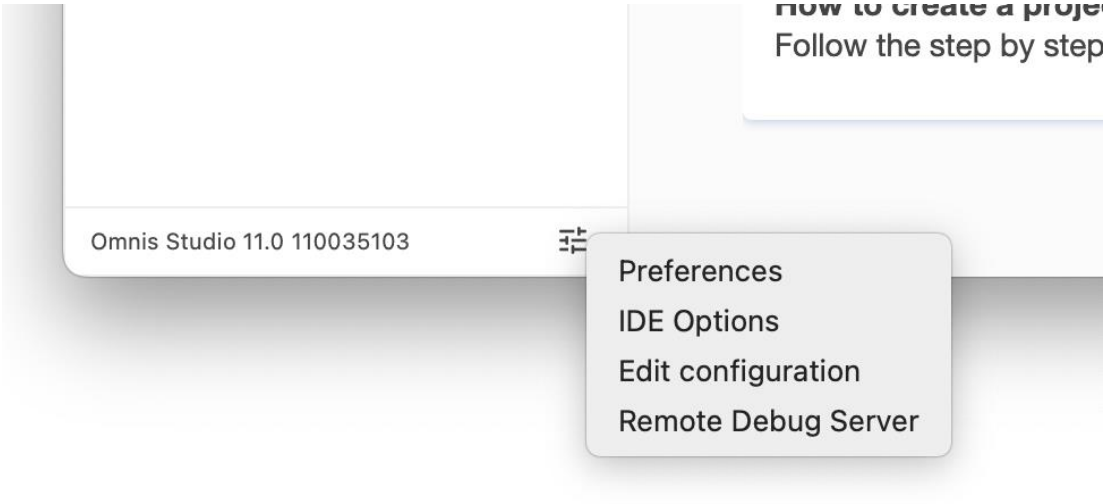
The enhancements in the Studio Browser include:

- The Libraries option in the Studio Browser is now called **Project Libraries** and this is the default option displayed when you start Omnis Studio (you can change this under the IDE Options in the Hub)
- You can create a new library using one of the options: **Web and Mobile, Desktop** (hidden in the Community edition) or **Blank.** The 'Web and Mobile' option creates a new library containing a NewRemoteForm and a Remote_Task, ready for you to start adding JS components, while the **Desktop** option creates a NewWindow for desktop use

- You can open an existing library using the **Open existing project library** button, plus the **Create project library from JSON** option allows you to import an Omnis library stored in JSON format
- The **Recent Project Libraries** area lists the libraries you have opened recently
- The tree list on the left contains the same options as before including the Hub, Remote Debug Client, the SQL Browser, the Omnis VCS, the Web Service Server, and the Trace Log
- The **Hub** option contains the same options as before, including Applets, Samples, Faults, and IDE Options. There are several **new example apps** under the Samples section, including a new category (at the top) called **'New'** to allow you to select all the new example apps in the last major release

The Omnis Studio version number was at the top of the Studio Browser tree list in previous versions, but this has been moved to the bottom left corner of the window.

The Omnis **Preferences** option has also been moved and it is now accessed via the Options button, next to the Studio version number. The Preferences option opens the Omnis root preferences ($root.$prefs) in the Property Manager.



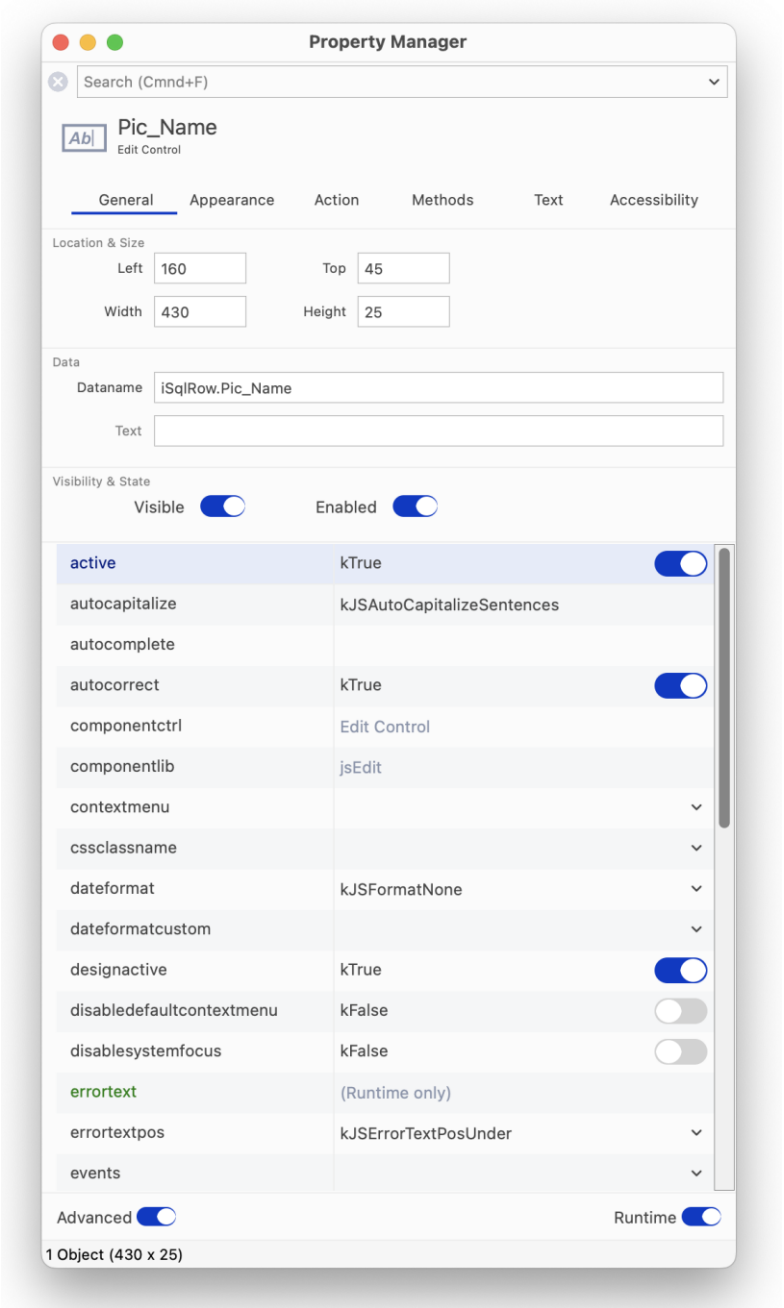
The **IDE Options** option opens the IDE Options tab in the Hub, which allows you to hide or show the main tools in the Studio Browser, as well as change the IDE theme.

The **Edit Configuration** option opens the new Configuration file editor; see later in this section.

The **Remote Debug Server** option allows you to configure and start the Remote debugger.

Property Manager

The **Property Manager** has been redesigned to make it easier to find and set properties. It now has a panel at the top containing the common properties for a component or object, including its object name, its location and size, its data properties (including $dataname), as well as the visibility and state properties ($visible and $enabled). You can use the Search box at the top to quickly find a property or subset of related properties, which is sometimes easier than looking under the tabs, e.g. enter color to find all color properties.



The 'Show All' option has been renamed 'Advanced' and is now placed at the bottom-left of the Property Manager window. The **Advanced** option is disabled (set to off) in new installations of Omnis Studio, so the Property Manager displays a simpler subset of properties for libraries, classes and components. In this mode, the tabs are hidden but you can right-click on a property to see which tab group it belongs to (see below). Existing users may like to enable the Advanced option to view all the properties for objects.

Property Tab

There is a new option on the Property Manager context menu that shows which tab the current property belongs to. (ST/CE/230).

The new option **Property Tab: <tab-name>** shows which tab a property is located on and is useful after a property search using the Search box at the top of the Property Manager. After a property name (or partial name) is searched, a subset of properties is shown in the Property Manager and all the tabs are hidden. You can Right-click on a property, select the **Property Tab: <tab-name>** option to jump to that property on the specified tab, ready to edit it. When there is no active search, the menu option is disabled, but still indicates the tab for the property.

Tab and Focus Selection

A few improvements have been made to navigating tabs and changing the focus in the Property Manager (and Catalog). (ST/HE/1451)

The Property Manager now tries to restore (by searching for name) the last tab that you selected (this is reset when closing and re-opening the Property Manager window), falling back to the previous behavior if the last tab selected cannot be retrieved.

In addition, you can now use the tab key to give the tabbed pane the focus, and then use the Left and Right arrow keys to switch tabs. You can tab out of the tabbed pane using the tab key, and in the Property Manager you can also do this using the Up or Down arrow key.

Copy Value

The **Copy Value** option has been added to Property Manager context menu to allow you to copy the value of a property, even if it is grayed out, such as when a class is not checked out of the VCS. (ST/VC/782)

Selecting Integer Values

You can now use the **Shift+Up** or **Shift+Down** Arrow keys to cycle through integer property values in the Property Manager, for example, when editing font sizes you can click into the property and use the Shift+Up/Down Arrow keys to increase or decrease the font size. (ST/HE/1763)

When increasing $fontsize in the Property Manager, labels and text objects in Remote forms and Window classes will increase their height if necessary, in order to correctly display a single line of text using the increased font size. (ST/HE/1764)

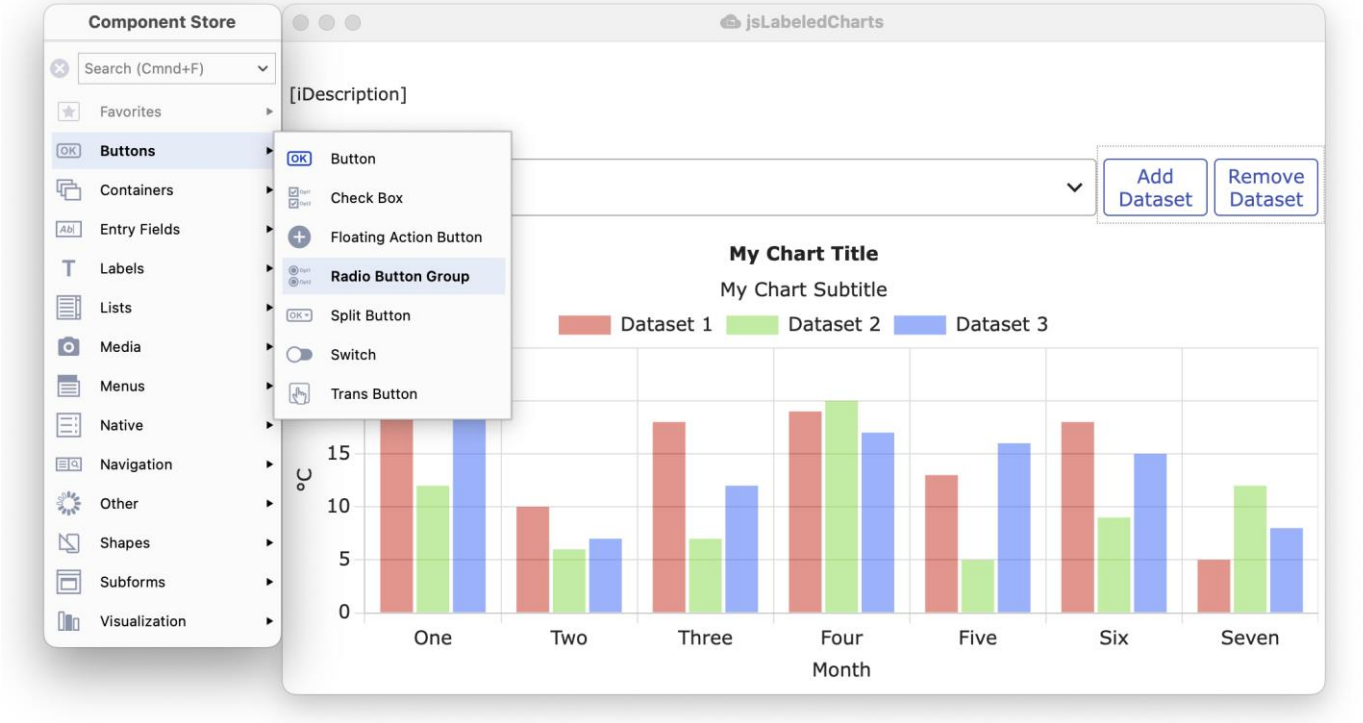
Check box list properties

When you click in a check list for a property, the changes are made to the property immediately. For example, when changing the properties of $gridlinesvisible in the check list in the Property Manager, the changes to the grid lines are seen immediately in the design form. The only exceptions to this new behavior are for $prefs.$idetools and $clib.$prefs.$designedscreensizes. (ST/JS/3030)

Component Store

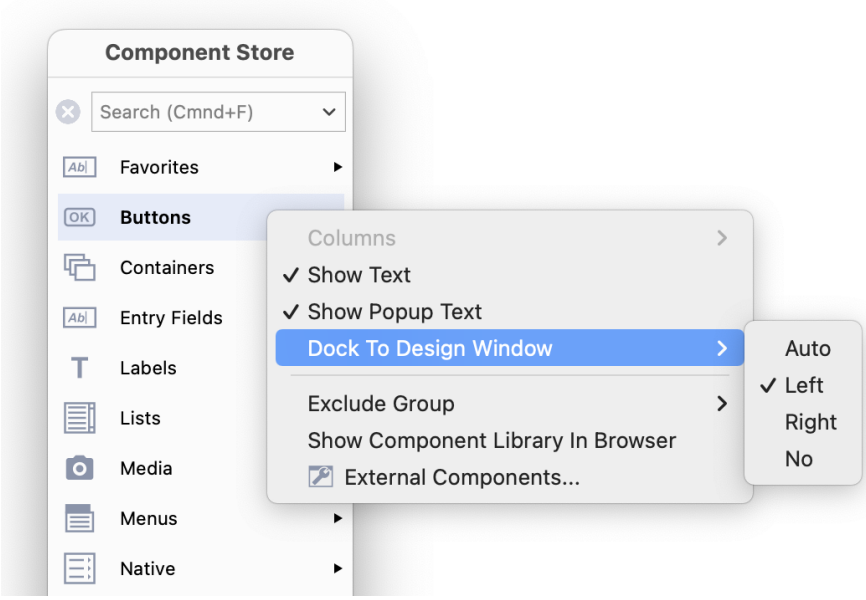
The Component Store has been redesigned and now appears in a vertical panel down the left side of the Remote form editor (and the editors for window, report & toolbar classes), with the components arranged in groups, such as **Buttons, Containers, Entry Fields,** and **Lists;** this provides a more compact layout than in previous versions where components were listed individually (note the old layout is no longer available and you cannot revert to the old layout).

When you open a class to edit it, such as a *Remote Form* class, the Component Store is opened automatically and, by default, it is docked to the left side of the design window. As you move a design window, the Component Store will remain docked and will move with it. The following screenshot shows a remote form in design mode and the Component Store is attached to the left side showing the **Buttons** group expanded.

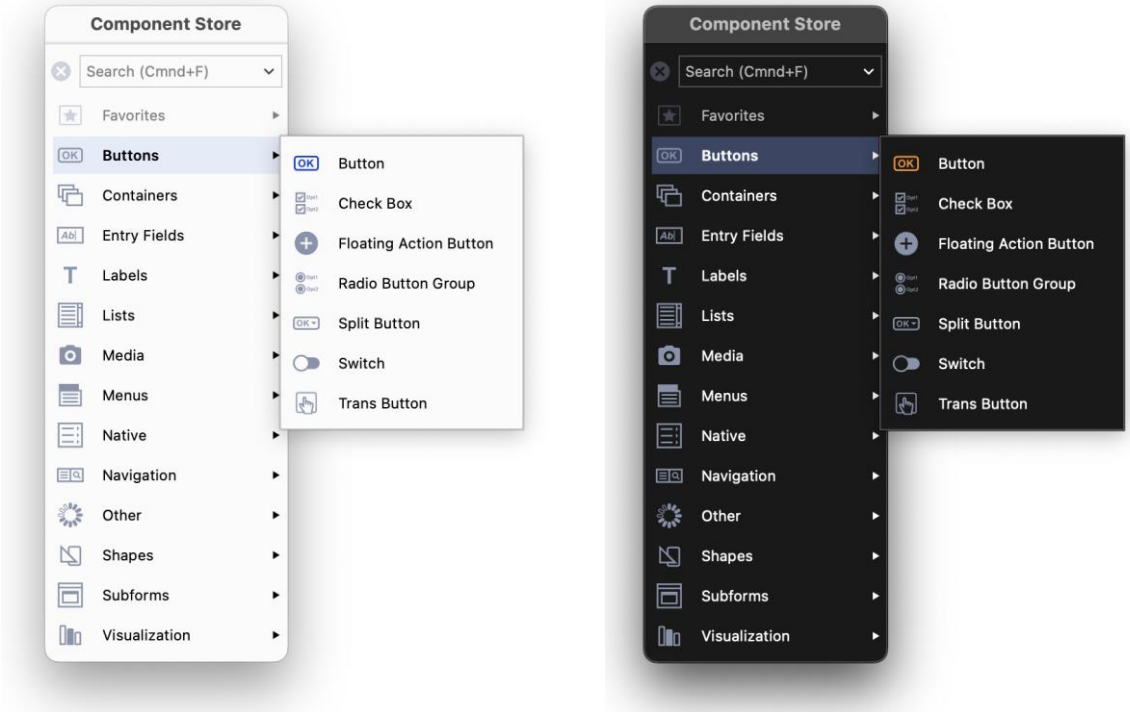


There are a number of components in each group, shown in the sub-menu that pops out when you click on a group, such as the **Buttons** group, which contains the standard **Button, Check Box, Floating Action Button** (a new component), **Radio Button,** and other types of button components. The Component Store is displayed using the current IDE theme, including default (light) and dark themes.

On the Component Store context menu, the **Dock to Design Window** allows you to set where the Component Store is docked, either Auto, Left (the default), Right, and No (not docked, but floating). In Auto mode, the Component Store will dock to the left side of a design window, but if there is insufficient space on the left, the Component Store will dock on the right.



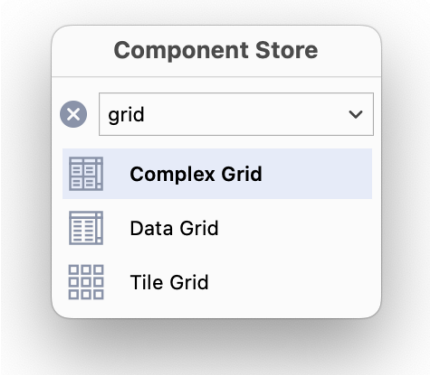
The contents of the Component Store is different for different classes, including Remote Forms, Reports, Window classes, or Toolbar classes. The following images show the **Buttons** group for a JavaScript Remote Form, using the default Studio IDE theme (on the left), and the same group(s) in Dark mode (on the right).



The initial view for the Component Store is to *Show Text labels* for the main groups and the components in the sub-menu popups, as shown above, but you can use its Context menu to change the appearance, e.g. hide the text or show it with 2 columns.

Searching for a component

You can use the **Search** box to locate a component or a group of similar components. As you type a search string, the contents of the Component Store is filtered, displaying only those components that *contain* the search string in their name, and the groups are hidden while the search is active. For example, you could enter "grid" to find all the grid components, as shown below. When the focus is on the Component Store, you can type **Ctrl/Cmnd-F** to put the focus into the Search box ready to type your search string.



Adding a Component to a form

To select a component, and add it to your Remote Form (or Report, Window, or Toolbar class), 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.

JavaScript Components & Groups

The following Component Groups and Sub-groups are available for JavaScript Remote form classes in the new Component Store (Window classes and Report classes have their own groups):

Group	JavaScript Component
Favorites	Right-click on a component & select 'Favorite' to add it to this group
Buttons	Button
	Check Box
	Floating Action Button NEW
	Radio Button Group
	Split Button
	Switch
	Trans Button
Containers	Paged Pane
	Scroll Box NEW
	Tab Pane (new compound object)
Entry Fields	Entry Field
	Rich Text Editor
Labels	Label
Lists	Combo Box
	Complex Grid
	Data Grid
	Droplist
	List
	Tile Grid NEW
	Tree List
Media	Camera NEW
	File Control
	Html Object
	Picture
	Video Player
Menus	Popup Menu
Native	Native List
	Native Slider
	Native Switch
Navigation	Hyperlink
	Navigation Bar
	Navigation Menu
	Page Selector
	Segmented Bar
	Tab Bar
	Toolbar

Group	JavaScript Component
Other	Activity
	Color Picker NEW
	Date Picker
	Device Interface
	Map
	Progress Bar
	Slider
	Timer
Shapes	Background Shape
Subforms	Subform
Visualization	Bar Chart
	Chart (Line, Area, etc) NEW
	Gauge NEW
	Pie Chart

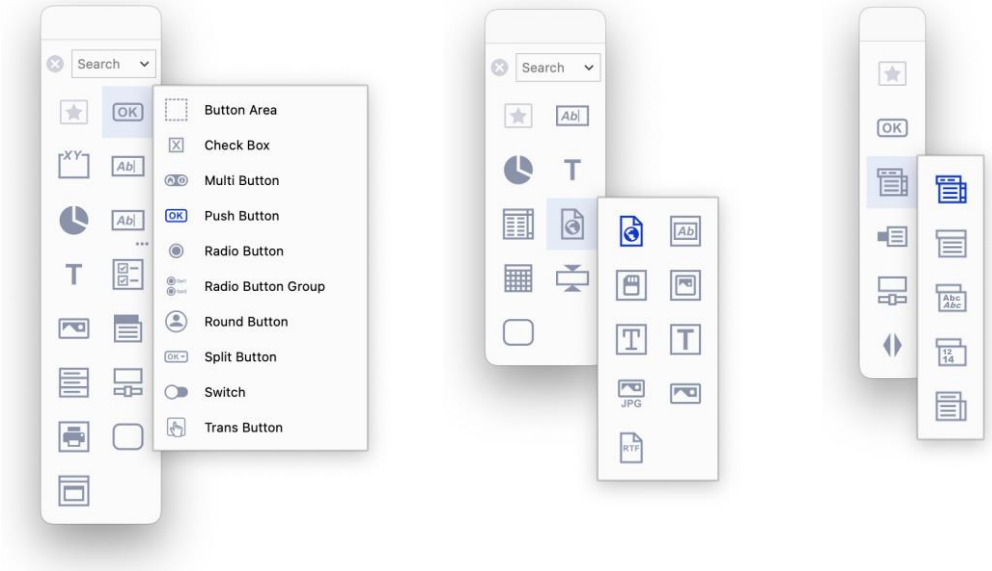
Existing users should note that the Device Control has been renamed to **Device Interface** and is located in the 'Other' group.

If you create any Compound Objects for Remote forms they will appear in their own group in the Component Store: you can define compound objects by editing the Component Store library; see below. For example, Remote forms have the Tab Pane in the Containers group, and Window classes have the Labeled Fields group containing the Labeled Entry Field and Labeled Masked Entry fields.

Window, Report & Toolbar Components

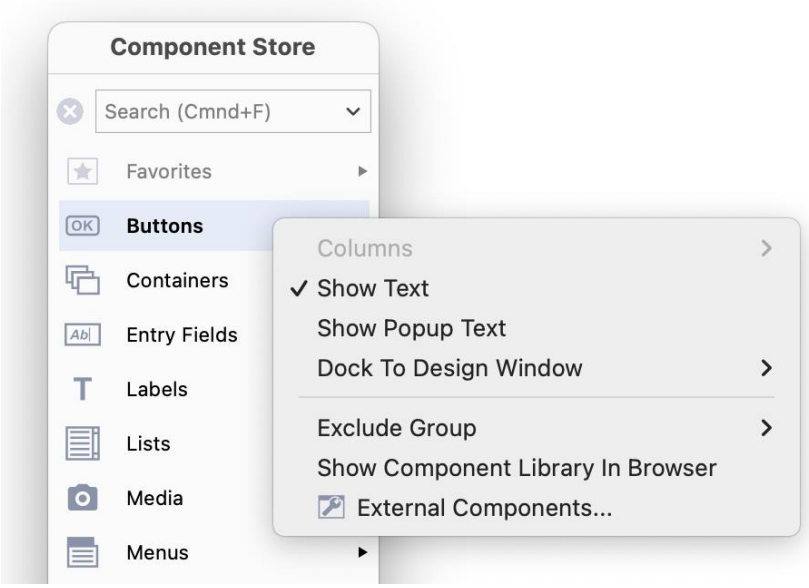
The new Component Store is used when editing a **Window, Report** or **Toolbar** class, and will contain components or groups relevant to the class being edited; all of the features described here apply to the Component Store for these class types.

Window class **Report class** **Toolbar class**



Changing the Appearance

You can change the appearance or layout of the Component Store using its Context menu. For example, you can Right-click/Ctrl-click anywhere on the Component Store and select or deselect the **Show Text** or **Show Popup Text** option to hide or show the text labels for the main groups or sub-menu popups, respectively.

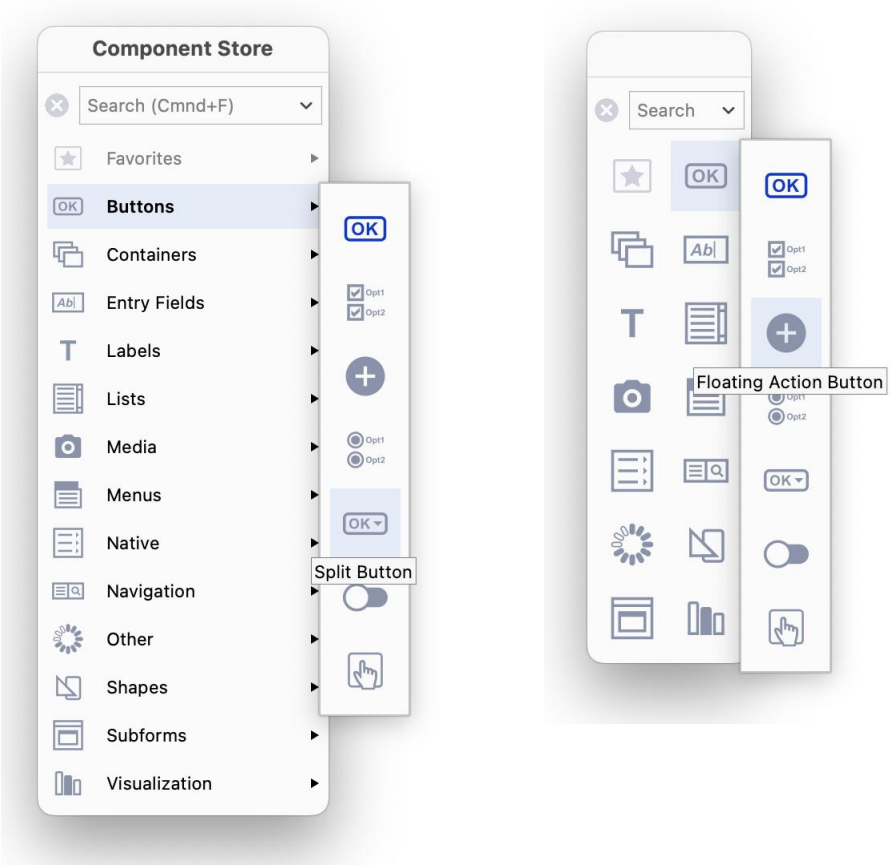


As you hide or show the Text labels, the icons will switch between *Large or Small* icons automatically (note the icons change automatically, so you cannot manually select large or small icons, as in previous versions). When the Text labels for the main groups or sub-groups are hidden, each icon has a tooltip that displays the component name or group name *as you hover over the icon*.

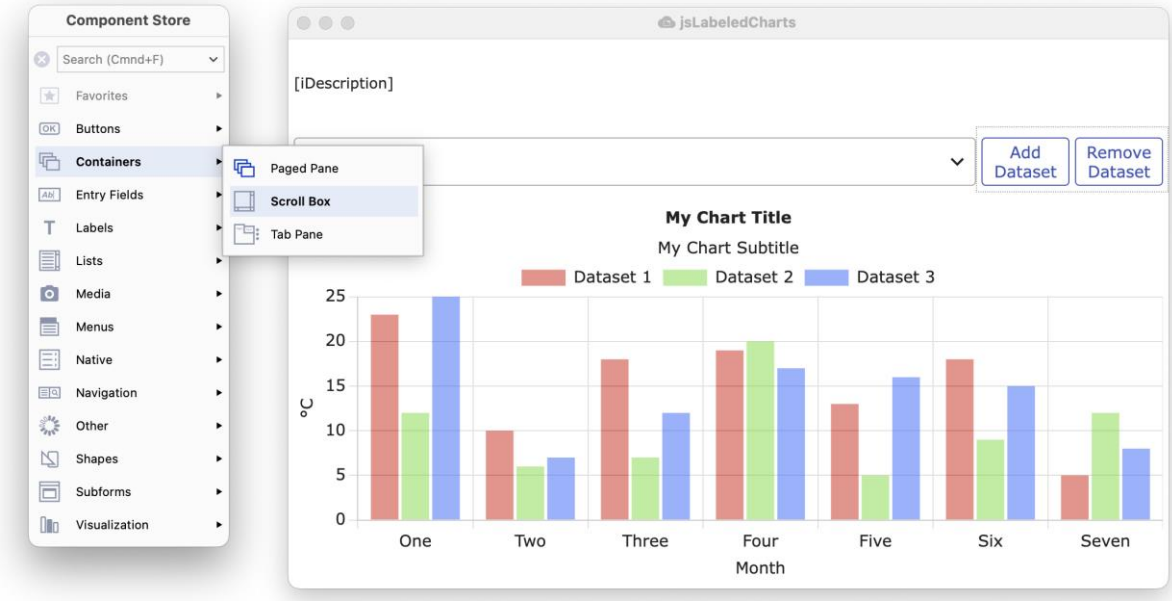
Note that there is no Save Window Setup command for the Component Store, since it saves its settings and current position automatically when it closes.

When the **Show Popup Text** option is disabled, tooltips are displayed on the components in a sub-group

When both **Show Text**… options are disabled, large icons are shown and tooltips are displayed on the main groups and sub-groups



If the **Dock To Design Window** option is set to Auto, the Component Store is docked or "attached" to the left side of the current design window, or if there is not enough space to the left of the design window the Component Store is docked to the right side of the design window. If this option is set to No (not docked), you can drag or "tear" the Component Store from the design window and it will float within the Omnis application window, plus its last position is remembered automatically. The following image shows the Component Store floating next to a remote form:



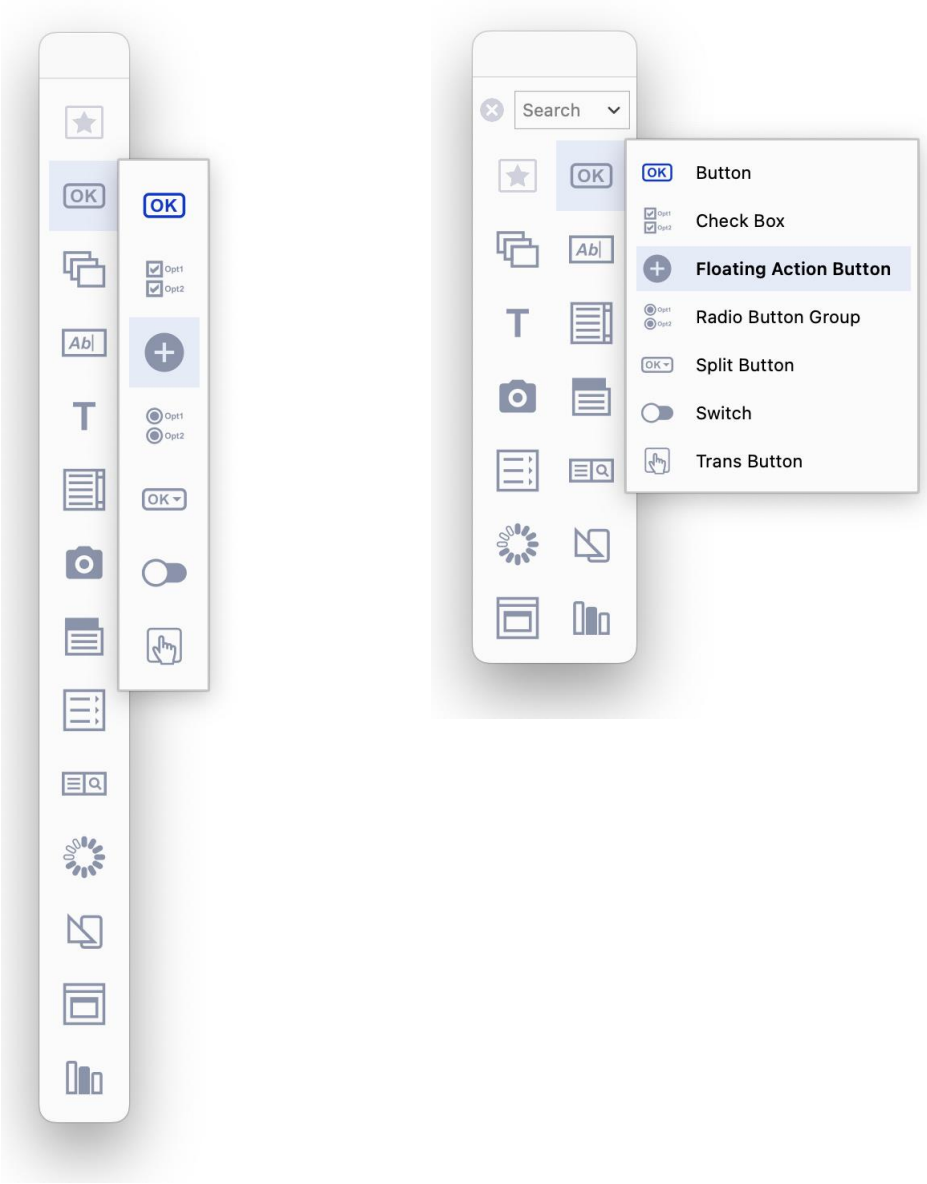
When the **Dock To Design Window** option is to Auto, Left or Right, you can temporarily drag or “tear” the Component Store away from the design window by dragging its title bar, but it will *snap back and become docked again* when you move or reopen the design window, or when you change the docking options from the context menu.

Two Column mode

When the Text labels are hidden on the main groups (i.e. the **Show Text** option is unchecked), you can configure the main group icons in 1 or 2 columns using the **Columns** options on the context menu (the sub-menu text can be enabled or disabled in 2-column mode, as shown below). The Columns option is disabled (grayed) when the Text labels on the main component groups are shown, and therefore you cannot enable 2-column mode in this case. Note the Search box is hidden when the Component Store is in single-column mode without text labels.

Single Column mode
Plus Show Text and Show Popup Text are disabled

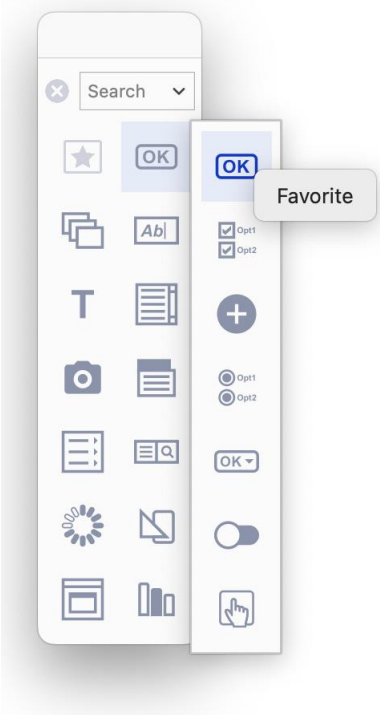
Two Column mode
Show Text is disabled,
Show popup text is enabled



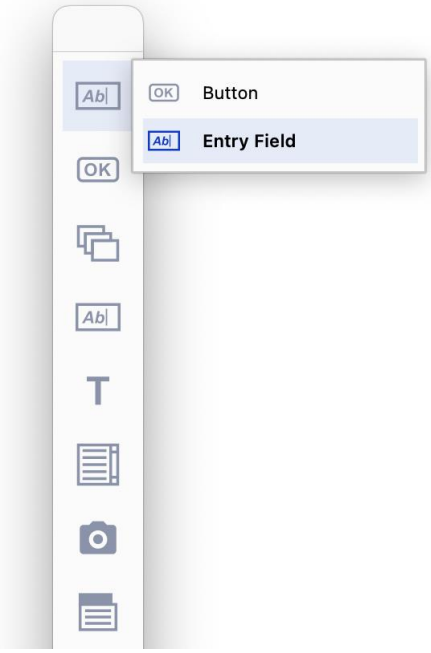
Favorites

You can add any single component to the **Favorites** group at the top of the Component Store window (shown initially with a Star icon), 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. For example, in the following screenshot the Button and Entry fields have been selected as favorites and are now shown together in the Favorites group at the top of the Component Store.

Right-click a component, Select **Favorite**; in this case, Button is added to the Favorites group



You can add components from different groups to the Favorites group; in this case, the Button and Entry Field have been added to the Favorites group



To remove a component from the Favorites group, you need to right-click the component and deselect the Favorite option.

Further Options

The options in the **Exclude Group** sub-menu in the Component Store context menu are checked by default, meaning that the **Deprecated** and **Internal** component groups are hidden or excluded by default; note that there are no Deprecated or Internal components for Remote forms, so these groups are only relevant for Window class controls at present. You are advised not to use the components in these groups, as they are included for backwards compatibility only, or for internal use, and should not be used for new applications.

The **Show Component Library In Browser** option allows you to change the contents of the Component Store and its groups; when selected, this option shows the Component Library (comps.lbs) in the Studio Browser, ready for you to edit it (as in previous versions). In general, you do not need to edit the Component Library, unless you want to add your own controls, compound objects, or class templates: see below.

The **External Components…** option opens the External Components dialog, allowing you to load external components (as in previous versions); this is only relevant for window and report classes, since all JavaScript components are loaded and displayed by default when designing remote forms. Note that all external components are shown

in the new Component Store even if they have not been marked in the External Components dialog to be loaded.

Configuration

There are a number of options in the Omnis Configuration (config.json) and Appearance (appearance.json) files that control the behavior or appearance of the Component Store. The time taken for a group sub-menu to pop out can be set using the **componentStorePopupDelay** item in the 'ide' section of config.json, an integer specifying the popup delay in milliseconds. The default is -1 meaning that Omnis calculates the delay to be just longer than the double click time, which means you can double click on an entry to add the corresponding default component to the design window without the popup appearing briefly.

There is a new 'componentStore' group in appearance.json containing the item **colorgroupdefault** that allows you to set the icon color for the default component in a group in the Component Store.

Editing the Component Store Library

IMPORTANT: *You are advised not to change the properties of any of the existing components or class templates, but to duplicate an existing control and make any changes to the copy. In most cases, you do not need to edit the Component Store Library, except if you want to create your own class templates or compound objects.*

The content of the new Component Store window is driven by the classes in the **Component Store** library called 'comps.lbs' (as in previous versions). To open the component library, select the **Show Component Library In Browser** option from the Component Store context menu, or you can Right-click on the **Libraries** node in the Studio Browser and select the **Show Comps.lbs** option (the latter is useful if you do not have a library open). The $componenttype property for all classes and templates that appear in the Component Store is set to kCompStoreDesignObjects.

All controls in the Component Store library now have the property **$componentinfo,** which is a row of information that specifies which group the object appears under in the new Component Store window. The $componentinfo property is visible in the Property Manager when you are editing a component on a Remote Form, e.g. in the JSFormComponents remote form class (also for Window, Report, or Toolbar classes).

Click on the $componentinfo property in the Property Manager to edit it: it has three columns defining the group, icon, and default status for the object:

- **group**
  The name of the group to which the object belongs. Group names are case insensitive, for example: Lists, Buttons, Entry fields.
- **iconid**
  The icon used for the object in the Component Store, which should be an SVG image placed in the new icon set 'componenticons'. Icons are displayed at 20x20 or 28x28 and SVG images will scale to fit the current size.
- **default**
  A Boolean that indicates if the object is the initial default in its group (the default object will change once a different object is chosen). If default is set to true for more than one object in the same group, the initial default will be the first object according to the case-insensitive ascending sort order of objects within the group by their name.

Compound Objects

A Compound Object is comprised of two or more standard objects grouped together to make a single object that appears in the Component Store, such as the **Tab Pane** for Remote forms (in the Containers group), or the **Labeled Entry Field** available for Windows Classes. When you drag a compound object from the Component Store, all objects in the grouped object are created in the remote form (or window).

You can create Compound Objects in the Component Store library, inside one of the Remote form or Window Component Store classes (or your own class but $componenttype must be set to kCompStoreDesignObjects). To create a Compound Object:

- Open the Component Store Library by right-clicking on the background of the Component Store and select the **Show Component Library In Browser** option; comps.lbs will be shown in the Studio Browser
- Open the Component Store class according to the type of Compound object (Remote form or Window class), then add the objects that will form the compound object, e.g. copy an Entry field and Label if you want to create a Labeled field; you are advised to create copies of the standard objects to form your compound objects. Note you cannot include line objects in a compound object
- Assign a name to the first object; this will be the name of the Compound object in the Component Store

For a Remote form, the first object is the object that occurs first in the field list window. For a Window class, the first object is either the first background object in the field list window, or if there are no background objects in the compound object, the first foreground object.

- Select all the objects that will form the Compound object, right click, and select **Group**
- Set the **$componentinfo** property of the group of objects, including the control name, group name and icon id (all members of the group should have the same value)
- Save the Component Store Library and close it

When the Component Store reloads in design mode, there will be a new Compound object with the specified name, group and icon id. The icon of a Compound object is shown in the Component Store with an additional … icon.

The dropped compound object has the same layout as its original objects, anchored at the top left of where you drop it.

You can use a responsive remote form to provide different layouts of the compound object for different breakpoints. Similarly, you can also set breakpoint-specific properties that will be set appropriately after dropping the compound object. Note that the Component Store Library may be using different breakpoints to your library, so the values used for each breakpoint in your library, after dropping a compound object, are the values for each nearest breakpoint, when comparing a Component Store breakpoint with a library breakpoint.

Container Compound Objects

The Component Store also allows you to create compound components using a Container field, such as a scrollbox or paged pane, with other objects inside the container. For example, you could create a compound object comprising a Scroll box (now available for remote forms) with a tab strip as its top toolbar component and a paged pane as its client component.

Class Templates

There are a number of Class Templates or Wizards that appear in the Studio Browser that are defined in the Component Store Library (such as Net Classes available in previous versions). You can create your own class templates, but the way you define these has changed (although the way you created class templates in previous versions is still supported for backwards compatibility).

Each class template in the Component Store Library has the new $componentinfo property, but for classes it has a single column named **group.** This allows a group to be specified for the class when it appears as a template or wizard in the Studio Browser. To use new $componentinfo property to define a class template:

- Select the class and set the group in $componentinfo for the class to the template name
- Set $componenticon to an Icon ID, preferably an SVG icon image
- Add a description for the class template in $desc

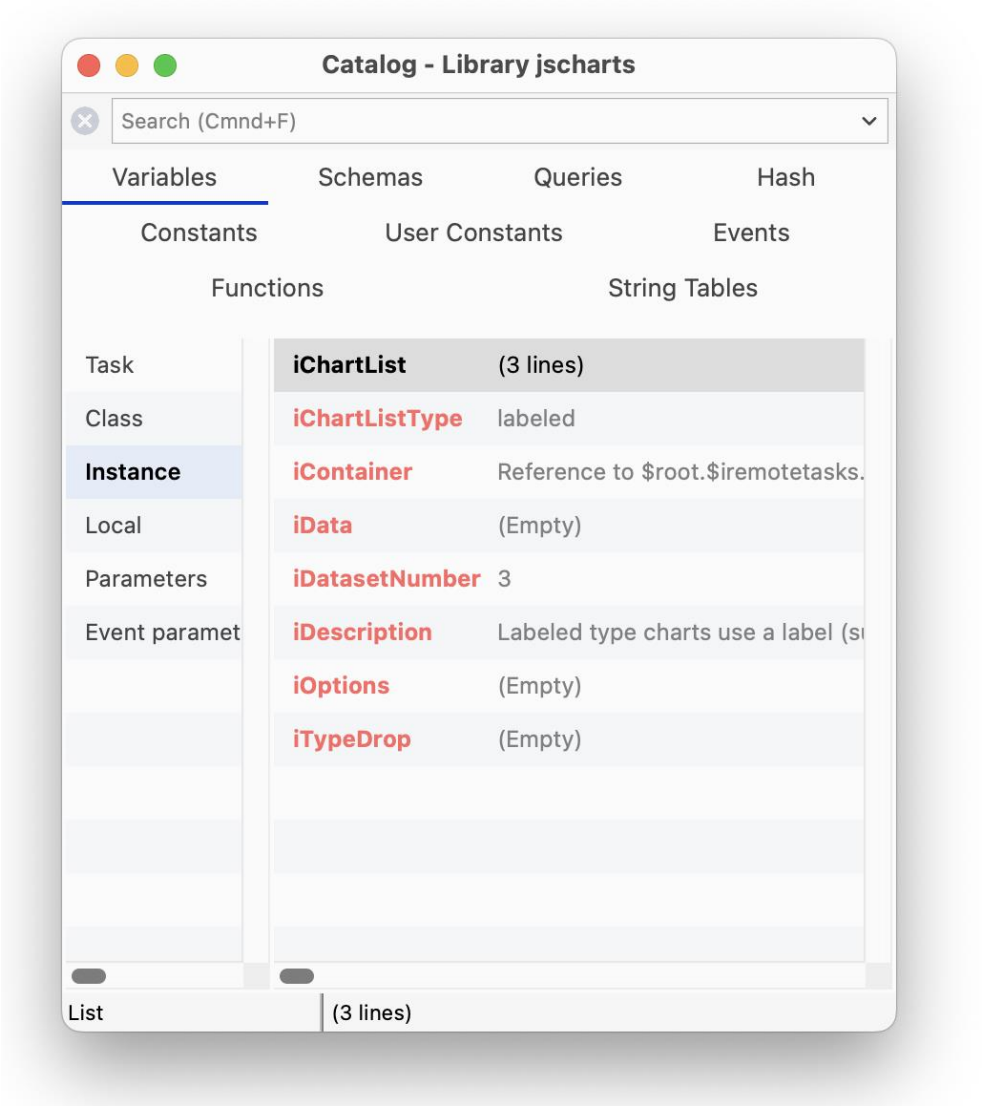
If you do not supply a group for a class template or wizard, it appears in a group named using its class type.

Catalog

Variable values

An extra column has been added to the **Catalog** window to show the values in the currently selected tab group, which will make it much easier to see the value of individual items or whole groups of items in one place. (ST/HE/1771)

The new values column is available for the Variables, Constants, Events, and Hash variable group tabs. For example, you can view the values for all instance variables under the Variables tab (assuming there is an instance open), as shown:



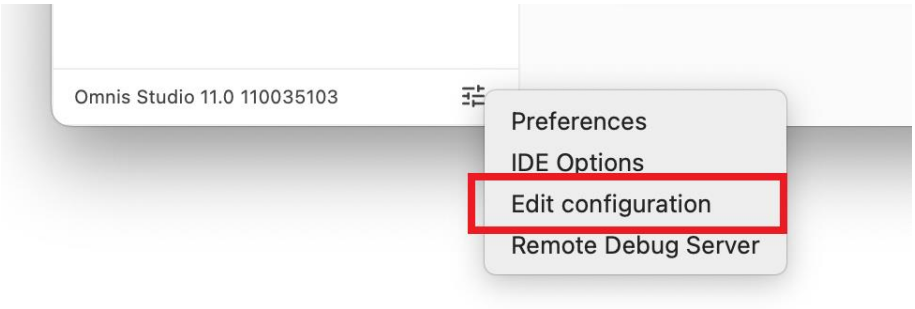
The values column is displayed as a third column on the right-hand side of the Catalog window, under each tab, and will show the current value of the variables or other items. There is a new option, **Show Values,** in the context menu for the Catalog that hides or shows the values column (default is on), which is saved with the **Save Window Setup** option.

Syntax Colors

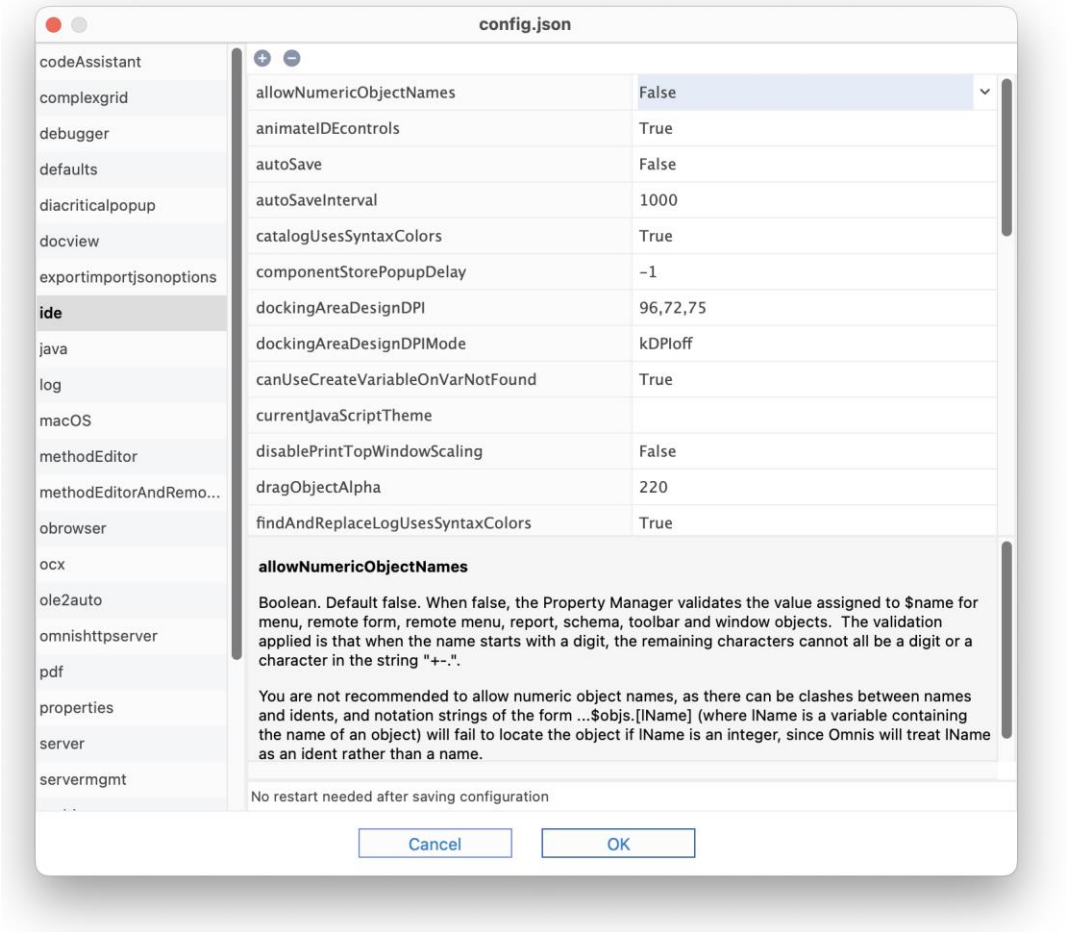
Items in the right hand list of the Catalog are now shown using the relevant syntax color, if any. There is a new item **catalogUsesSyntaxColors** in the 'ide' section of config.json, that can be used to control this behavior; the default is true.

Configuration File Editor

There is a new editor to allow you to edit the settings in the **Omnis Configuration** file (config.json) inside Omnis Studio, rather than having to edit the file in a text editor (although you can still do this). To edit the Omnis Configuration file using the new editor, click on the Options button at the bottom-left of the Studio Browser, then select the **Edit Configuration** option.



The Configuration Editor shows the main groups of items in the config.json file in the left hand list, such as ‘defaults’, ‘ide’, and ‘methodEditor’, and for each selected group the items within that group are shown on the right, for example, the **ide** group of items is shown below:



Some items require a string value, in which case you can click on the item and edit it directly in the text field, otherwise, when you click an item to edit it, a droplist may appear containing its possible values (such as True/False values), or some other kind of dialog will open, such as a file select dialog or a color picker.

When an item is selected, the **Help** panel below the configuration items grid provides a full description of the item. In addition, the status bar beneath the help panel indicates whether or not a restart is required after changing the item and saving the configuration file. The status bar is empty when the item is not relevant to the current platform.

There is a complete list of items and groups allowed in the Configuration file in the Appendix at the end of this document.

Adding Configuration items

The + and - icons at the top of the window allows you to add or remove items, in general however, ***you should not delete items,*** rather just change their values; for example, for a Boolean item value which you want to disable, set the value to False to disable it rather than deleting the item.

To add an item, click on the **+** icon, enter the name of the Config item, and choose the *type,* which is one the following types:

- **Boolean**
  a True or False value
- **Character**
  a string
- **Integer**
  an integer value (usually in a specific range of values), or a constant value
- **List**
  For list items, enter the item name, then you can specify the list of items in a popup window; they are displayed as a comma-separated list

In addition, you can now enter \t to mean tab, e.g. for log.conversionLogDelimiter.

There may be specific items that are included in the documentation or provided by Technical Support, that are not included in the default Configuration file, which you can add using the Configuration file editor.

Updated or Unsupported Items

The naming of the items in the Configuration file has been tightened up for this version, and as a consequence a few configuration items have been renamed or changed:

- jvm in the java group is now **jvmPath**
- python in the windows group is now **pythonPath**
- iconsFolder in the server group is now **iconsFolderName**
- **miniconid** in the windows group is now a character string (to allow for SVG as well as PNG)
- diacriticalpopupuseosxkeyboardlayout in the diacriticalpopup group is now **diacriticalPopupUsesMacOSKeyboardLayout**

The following configuration items are no longer supported:

- **blankLinesToAdd** and **maxWidthOfMethodTooltip** in the methodEditor group
- **candebug** in the obrowser group
- **autoChunkRESTfulURLs** in the server group

Configuration Editor Visibility

The Configuration file editor is available in the Development version of Omnis Studio, as well as the Runtime and Server versions (but not the Linux Headless server). To open the Configuration file editor in the Runtime or Server version, select the **Edit Configuration…** option from the **File** menu. You can hide this option in the Runtime or Server version by executing the sys(246) function, or sys(247) will show it again; the default setting is for it to be visible.

Help Files

The contents for the Help provided for the Configuration items are stored as HTML pages in a new folder called 'confighelp' in the Studio folder; this folder is not present in the Headless Server version.

Spell Checking

Omnis now checks spelling in text in end user apps, as it is entered in desktop forms (on the fat client), and in the Studio IDE during development; note this does not apply to JavaScript client remote forms. Spell checking allows words to be validated, based on the local language setting, and spelling suggestions are presented in the UI or used automatically, including the highlighting of misspelled words, and correcting misspelled words as they are entered.

Support for spell checking is provided by calling the *Spell Checker API* on the current operating system, including under Windows and macOS. Spell checking is enabled by default and will be used in the right context automatically, such as in Entry fields or in the Code editor, and there are various options or settings in the Studio IDE to manage spell checking.

Configuration

There are two options for how Omnis chooses a language to use with the system Spell Checker APIs. Which of these two applies depends on the entry **useSystemSettingsForSpelling** in the 'defaults' section of the Omnis Configuration file (config.json).

If useSystemSettingsForSpelling is true (the default), Entry fields use the system settings to identify the current language or languages. For macOS, this means the settings in the Keyboard, Text panel in System Preferences. For Windows, this means the System Locale.

If useSystemSettingsForSpelling is false, Entry fields use the National sort ordering locale for the current language in the Omnis localization data file.

If Omnis fails to initialize the system Spell Checker API to use the required language it reports this failure to the Trace log.

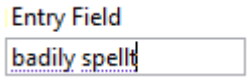
Window Class Controls

The following Window class (fat client) controls allow spell checking: **Single-** and **Multi-line Entry field, Combo box, String grid,** and **Data grid.** These controls have the following properties to control spell checking:

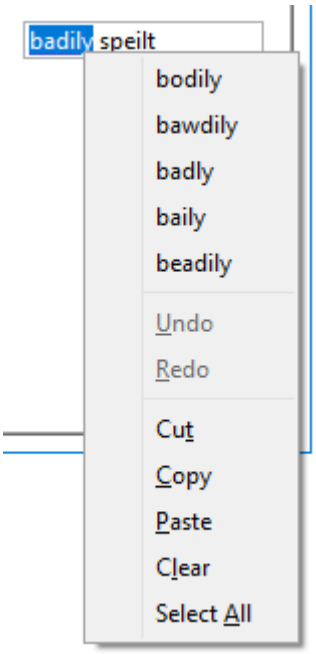
Property	Description
$showspellingerrors	If true, the control underlines spelling errors using a dotted line
$autocorrectspelling	If true, and the user types a separator (e.g. space or comma) when no text is selected, the control replaces a misspelled word immediately before the selection with a correctly spelt word. Note that Undo allows you to revert to the originally entered text, and then continue typing without correcting it again

These properties are kFalse in existing or converted apps to maintain previous behavior.

The dotted line used to underline spelling errors uses the new color "colorspellingerror" in the system (and system.dark) section of appearance.json. The following screenshot shows an Entry field containing misspelled words:



When $showspellingerrors is true, and the currently selected text in an Entry field is a misspelled word, the default context menu for the edit field includes up to 10 spelling suggestions, before the normal menu commands, such as Cut, Copy and Paste. Selecting one of these suggestions from the menu replaces the currently selected word.



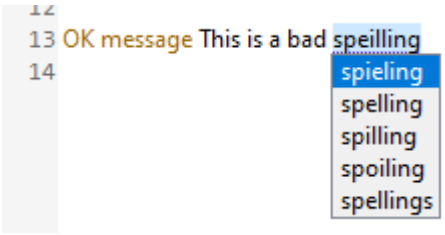
Code Editor

Spell checking is also enabled in the Code Editor (Method editor); there is a new **Show Spelling Errors** option in the View menu that is enabled by default.

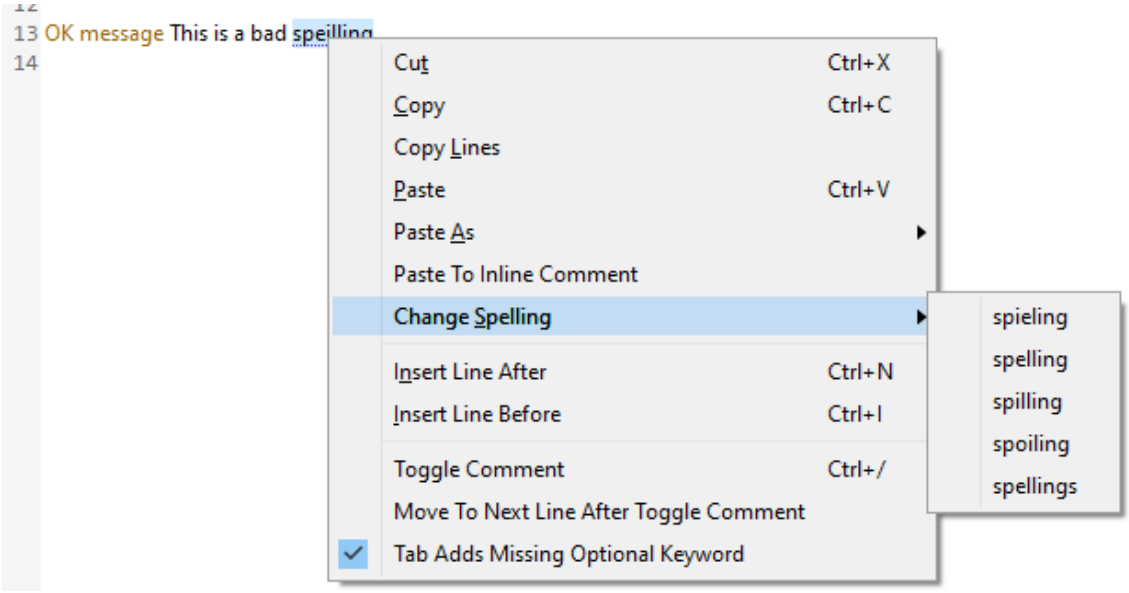
Misspelled words in strings entered into code are underlined in the same way as edit fields underline spelling errors when $showspellingerrors is true. In addition, misspelled words outside *Square Brackets* are underlined for certain commands, including OK message, Yes/No message, No/Yes message, Prompt for input, Text:, Line:, and Send to trace log.

In addition, when **Show Spelling Errors** is enabled in the Code Editor, you can change the spelling of a *selected word* (which need not be misspelled) in either of two ways, described below (this applies to a string or outside square brackets in the commands as listed above).

You can select a word, and from the **Modify>>Selection** submenu you can select the **Change Spelling…** option: note that this command is only present if there are some possible suggestions for the selected word. You can also use the keyboard shortcut **Ctrl/Cmd+B** to change a word (specified in the **changeSpelling** key in the methodEditorAndRemoteDebugger section of keys.json). After selecting the command, a popup appears from which an alternative spelling can be selected to replace the word.



Alternatively, you can select a word, Right-click on it, and the context menu contains a new **Change Spelling** hierarchical menu, with up to 10 suggestions, that can be used to replace the selected word.

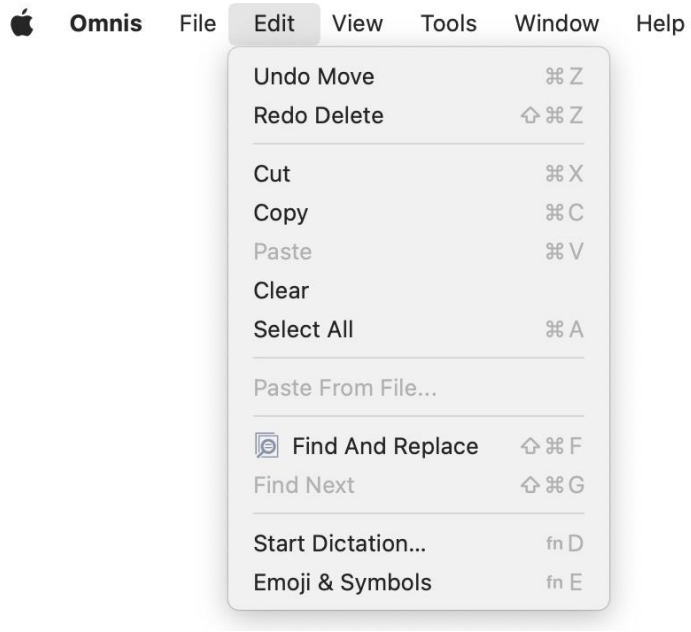


Remote Debugger

The Remote Debugger also supports spell checking, enabled using the **Show Spelling Errors** option. When this is true, for an edit session, the context menu for the editor includes up to 10 suggestions when the selected text is a misspelled word (that is, it behaves like a normal Entry field with $showspellingerrors set to true).

Multi- Undo and Redo

Omnis now supports multiple **Undo** and **Redo** operations in the class design editors and the Method Editor. Omnis stores most operations on an *Undo* and *Redo Stack* which can be accessed using the **Undo** or **Redo** commands in the **Edit** menu, or using **Ctrl-Z** or **Ctrl-Y** key strokes on Windows, or **Cmnd-Z** or **Shift-Cmnd-Z** on macOS.



As you undo and redo operations in a class editor, or the method editor, the **Undo** and **Redo** commands will update in the Edit menu to reflect the next operation that can be undone or redone. When there are no operations that can be undone or redone the corresponding option in the Edit menu will be grayed out.

In general, most operations that support (single) Undo support multiple Undo and Redo, including moving and resizing objects, adding and deleting controls (including Cut and Paste), object property changes (in the Property Manager), align menu operations, and changing or deleting layout breakpoints in remote forms.

In effect, a separate Undo stack is kept for each editor, so as you switch from one editor to another, e.g. between two remote forms, the Undo or Redo commands will apply to the stack for that class editor (this does not apply when opening the Method Editor, see below). There is currently no limit on the number of operations that can be stored on the Undo stack.

To enable multiple Undo and Redo, Omnis saves a copy of the class data before and after an operation. To support this, there is a new temporary folder named 'undotemp' created automatically in the 'studio' folder at startup, which contains temporary copies of class data associated with undo stack entries; these files are deleted automatically, but in case they are not, any stray files are deleted when Omnis starts up.

Property Manager

You can Undo a property change when the Property Manager has the focus, provided that the current line in the Property Manager does not itself have an Undo stack (this can apply when the edit field has the focus). When you undo a property change, Omnis tries to select the affected property in the Property Manager. Undo works for inheriting and overloading a property.

Method Editor

If you open the method editor for a class, while the design editor for the class is open, Omnis clears the undo stack of the class design editor (but only if something is changed in the method editor). This prevents Undo or Redo in the class editor overwriting the class and losing any method changes.

Report Editor

Undo works in the report editor for the following operations: moving a report section, inserting or deleting a report line, and editing the page setup. Note that the report editor does not support Undo or Redo for the sort fields dialog. When you open this dialog, Omnis clears the report editor undo and redo stacks.

Form or Window Editor

Most operations within complex objects, such as a Complex grid or a Tab strip, support multi- Undo and Redo, such as, setting column widths in a Complex grid using the mouse or changing a grid line property.

Appearance Color Format

The syntax used for colors in appearance.json has been changed to make the file easier to use and read. The syntax changes occur automatically, so if you load appearance.json using the old syntax, Omnis replaces the content with the new syntax (and writes the file back to disk).

Colors must now be a string, which can be either “#RRGGBB” or “kColorDefault” or one of the 16 standard colors: kBlack, kDarkBlue, kDarkGreen, kDarkCyan, kDarkRed, kDarkMagenta, kDarkYellow, kDarkGray, kBlue, kGreen, kCyan, kRed, kMagenta, kYellow, kGray, or kWhite.

To see the different format, compare the old format:

```
"IDEgeneral": {
        "clientexeccolor": -2147483599,
        "colorpropertymanager": 16448250,
        "colorreportdesignposnsectiontext": -2147483599,
        …
    },
```

With the new format:

```
"IDEgeneral": {
        "clientexeccolor": "kColorDefault",
        "colorpropertymanager": "#FFFFFF",
        "colorreportdesignposnsectiontext": "kColorDefault",
        ...
    },
```

You can extract a color from appearance.json using the following code:

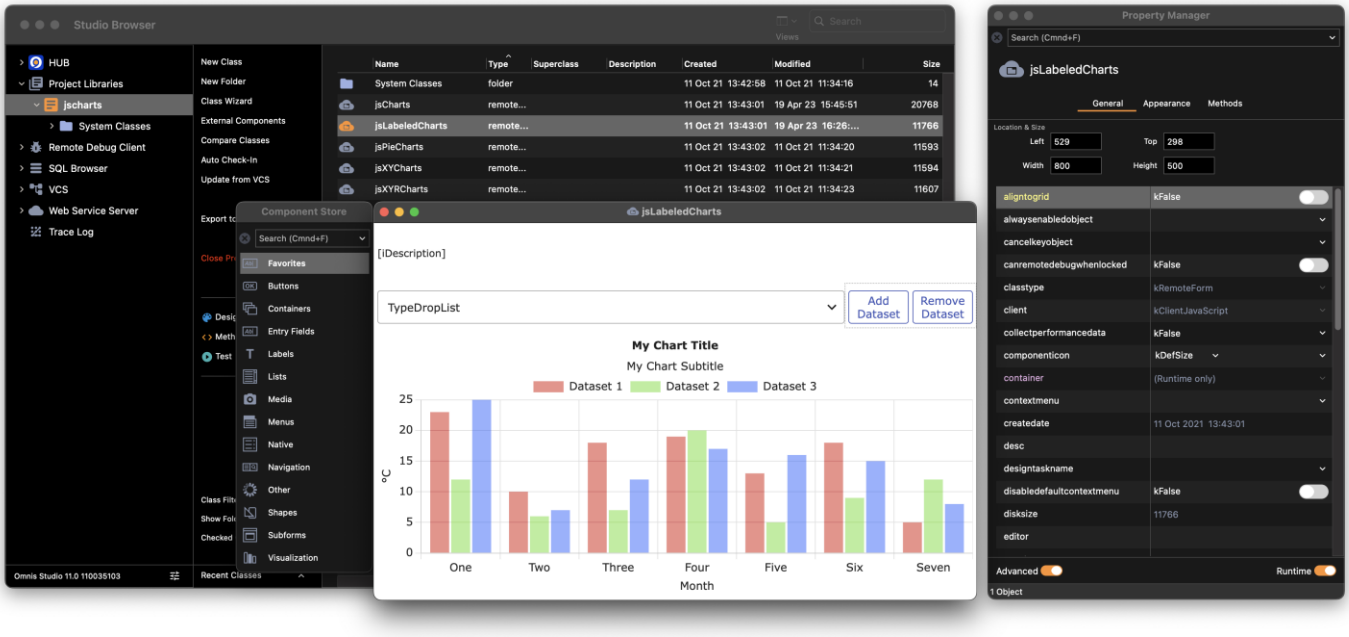
```
Calculate lAppearanceJSON as $prefs.$appearance
Calculate lColorMethodLines as lAppearanceJSON.compareTool
Calculate #S1 as lColorMethodLines.comparemethodlinescolor
If left(#S1,1)='#'
    Calculate #1 as hexcolor(mid(#S1,2))
Else
    Calculate #1 as [#S1]
End If
```

This assumes that the entries are either #RRGGBB or a constant such as kColorDefault. This code could be wrapped into a method such as $getappearancecolor(“group”,”color”) returning the resulting color.

Dark Mode

Support for **Dark Mode** has been enabled on all platforms (ST/HI/1909). Dark mode is supported on macOS 10.14 and later or Windows 10/11 or above. You can change the system color mode via the **System Preferences > General** option on macOS, or the **Settings > Personalization > Colors** option under Windows.

Omnis Studio supports switching between dark and light modes when using *the default Omnis design theme* (studio/themes/appthemedefault.json); the theme for design mode in Omnis is set via the **Themes** tab under the **IDE Options** option in the Studio Browser (via the Options button at the bottom-left). The colors in the default design theme have been updated to work with Dark mode on macOS and Windows.



Dark Mode in Themes

Support for dark mode has been enabled by adding items named "[item].dark" to any of the theme file, as well as the main appearance file (appearance.json in the omnis\studio folder). For example, as well as a "tree" item in appearance.json, there is a "tree.dark" item which is used when the system is in dark mode; if there is no ".dark" entry, the normal entry is used in dark mode.

If an appearance.json file does not contain any ".dark" entries, Omnis will use the light system theme when determining any defaults that come from the system, although system dialogs will display in the current mode for the system.

User Defined Colors

User defined colors can be added to the appearance.json which can be used for theme colors for window class controls in desktop (fat client) apps.

The new colors replace the 16 basic colors at the bottom left of the standard palette of the color picker, used when specifying the color for window class controls. They are defined using the groups "user" and "user.dark" in the appearance.json theme file, using the names **color1** to **color16.** The defaults for these correspond to the 16 basic colors in the color picker available in previous versions.

They are represented by 16 new color constants **kColorUser1** to **kColorUser16.**

IDE Window Colors (Windows only)

You can now specify dark mode colors for some of the IDE window colors defined in the $windowoptions Omnis preference; these only apply on Windows OS and are used automatically when dark mode is being used. The following colors can be defined:

titleactivecolor.dark
titleinactivecolor.dark
smalltitleactivecolor.dark
smalltitleinactivecolor.dark
borderactivecolor.dark
borderinactivecolor.dark
captionactivecolor.dark
captioninactivecolor.dark
smallcaptionactivecolor.dark
smallcaptioninactivecolor.dark
minmaxbuttonhotcolor.dark
minmaxbuttonhottrackingcolor.dark
closebuttonhotcolor.dark
closebuttonhottrackingcolor.dark

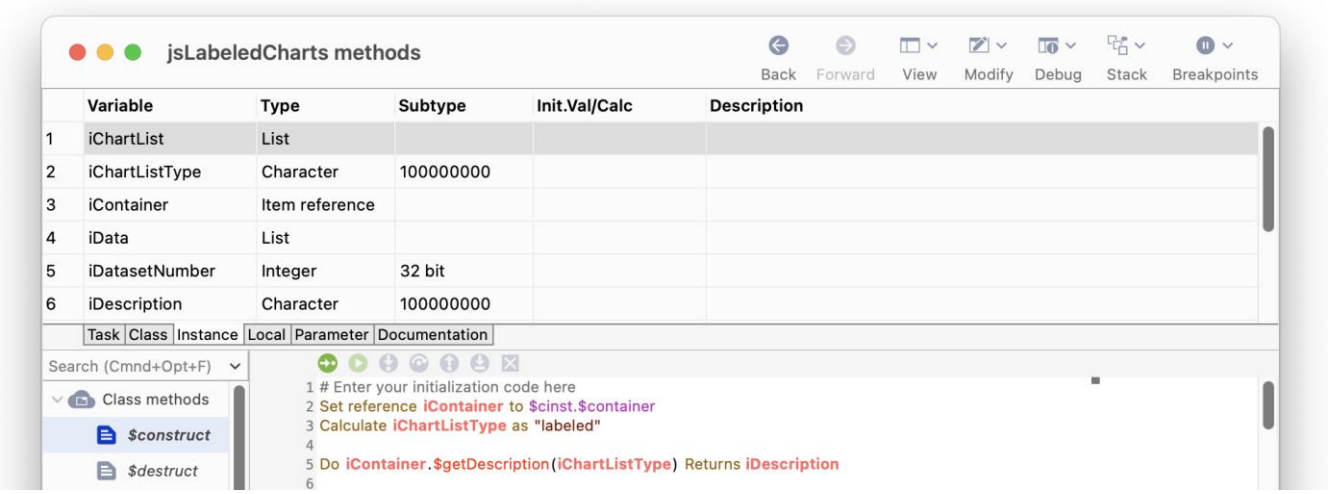
System Colors

The system colors **colorqbackfill, colorqforefill** and **colorqframe** have been removed from appearance.json (these correspond to internal color constants that always need to map to the same color, and therefore should not be customized). (ST/RC/1404)

Design Window Titles

The titles displayed on class editor windows and other tools, including the Method Editor, have been modified so that class names are visible on macOS Big Sur (or above); in testing of this on macOS we found class names were not visible so the format of design window titles has been redesigned. In addition, some of the debugger commands have been moved. (ST/HE/1765)

Class design window titles now display just the ***class name,*** or for classes that have methods, the class name followed by the word “methods” is displayed, for example, for a remote form class named jsLabeledChart, the two titles used are “jsLabeledChart” and “jsLabeledChart methods” (see below):



For an object class named oObject, the title used for its methods is “oObject”. Note also that on macOS the toolbar options (View, Modify, Debug, etc) are incorporated into the window title bar.

When more than one library is open (in the Studio Browser), the title is prefixed with the ***library name,*** for example, “myLibrary.wWindow”. As you open and close libraries in the Studio Browser, the titles update as necessary, to add or remove the library name prefix. For system class editors, some useful text is displayed rather than the class name, e.g. “Text formats” (instead of #TFORMS).

Find and Replace

Find Matches

Find matches are now underlined in the Find and Replace log, rather than filling the rectangle as in previous versions. This change applies to the Find and Replace log when the Highlight Matches option in the log context menu is enabled (the default). (ST/DB/1340)

Recent Search List

The maximum number of searches saved in the Recent Search drop list in the Find and Replace window has been increased to 30. (ST/FR/147)

All droplists and combo boxes in the IDE now use a configuration item **maxDisplayedDropListLines** in the 'ide' section of config.json to specify their maximum number of displayed lines. This defaults to 30, and can be 5-50 inclusive.

Checked Out Classes

The **Show Checked Out Classes In Log** option has been added to Find and Replace log context menu to allow you to show which classes in the Find and Replace log are checked out of the VCS; the option is enabled by default and is saved in Window Setup. Changing the option via the context menu does not cause lines already in the log to be updated. (ST/DB/1401)

$findandreplace method

The **bReturnLog** parameter has been added to the $findandreplace() class method, which provides an alternative to calling sys(241) to return the Find and Replace log. (ST/FR/163)

The definition of the method is now:

- **$findandreplace**(cFind, cRep [,bIgnCase=kTrue, bWholeWord=kFalse, bRegExp=kFalse, bClearLog=kFalse, **bReturnLog**=kFalse])

If cRep is #NULL, the method finds all instances of cFind, otherwise, replaces all instances of cFind with cRep. Returns status row.

When bReturnLog is kTrue (the default is kFalse), the status row has an additional column named **Log** that contains the Find and Replace log (this has the same structure as the list returned by sys(241)).

Trace Log

Styled Text

The Trace log now allows text styles to be added to the logged text. (ST/DB/1410)

The *Send to trace log* command supports text styles, added using the *style()* function inside square brackets, such as kEscColor and kEscStyle. For example, you could apply colors to sections of the logged text when it is displayed in the trace log panel in the browser or the trace log window; such styles are stripped when writing the trace log line to the text log file in the logs folder.

The trace log renders the text styles if the new entry **traceLogUsesStyles** in the 'defaults' section of config.json is set to true; this replaces the ide entry traceLogUsesSyntaxColors, which has been removed.

Note that if you use styles other than kEscColor and kEscStyle, these styles are ignored when copying selected trace log lines to the clipboard as HTML.

For JavaScript client-executed methods, where the *Send to trace log* command sends the text to the JavaScript console (if available), text styles are not supported.

Log Font Size

The font size for the Trace log (and the Find and Replace log) is now saved in the window setup. (ST/DB/1339)

The **Save Window Setup** option saves the current font size for the Trace log, and the Find and Replace log. Note the font size of the trace log panel in the browser is not saved.

Using Multiple Screens on macOS

A new option has been added on macOS only that allows you to move the top window to an additional screen. (ST/HE/1603)

The **Move Top To <screen>** command has been added to the Window menu, on macOS only (10.15 Catalina and later) allowing you to move the top window to the named additional screen, including design tools such as the Property Manager and Method Editor, or any class editor window. The new command will only appear when there is more than one screen connected to your Mac computer.

Tooltips

There is a new entry "tooltipfontsize" in the “tooltip” section of the appearance.json theme file to set the font size for tooltips. Set it to zero to use the system default, or 6-32 inclusive. For macOS only, there is a new entry "ThemeTooltip" which is the font used for tooltips.

Tooltips now use the non-system style of tooltip. There is a new entry in the “tooltip” section of appearance.json, named “hidearrows” which defaults to false. When true, the arrow for a non-system style tooltip is not displayed.

Single Instance Preference

There is a new option ‘singleInstance’ in the ‘windows’ section of the config.json file that can be used to set the value of the Omnis preference $prefs.$singleinstance. When set to true (the default is false), the value of the $prefs.$singleinstance property is set to kTrue and only one instance of Omnis Studio is allowed.

JavaScript Components

JS Chart

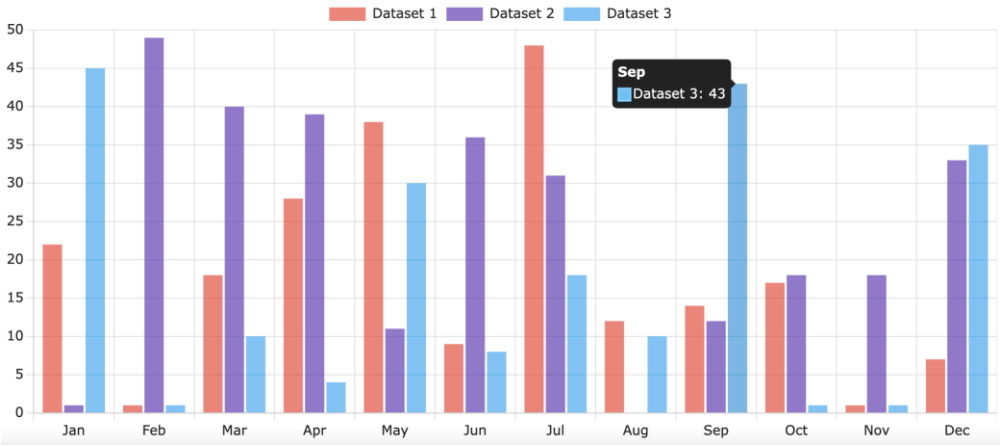
The **Chart** is a new JS component that allows you to create different types of chart from list data to display in a remote form. It uses the Chart.js JavaScript library, an open source library available under the MIT license, which you can use in your applications (with the correct license attribution). The new JS Chart control provides you with a wider range of chart types than the existing Bar chart and Pie chart components, and provides a more 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.

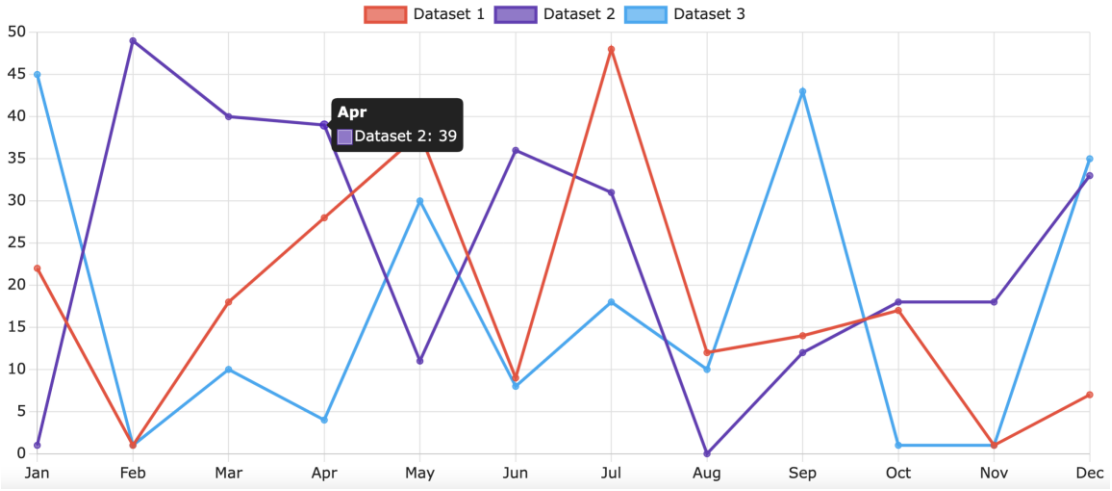
Chart type	Description
Line **Bar** **Radar**	Line, Bar, and Radar type charts (or Labelled charts) use a ***label*** (e.g. a month) for the X axis (horizontal), and a ***value*** for the Y axis (vertical).
Pie **Doughnut** **PolarArea**	Pie, Doughnut, PolarArea charts (or Area type charts) use the same 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).
Scatter **Bubble**	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.

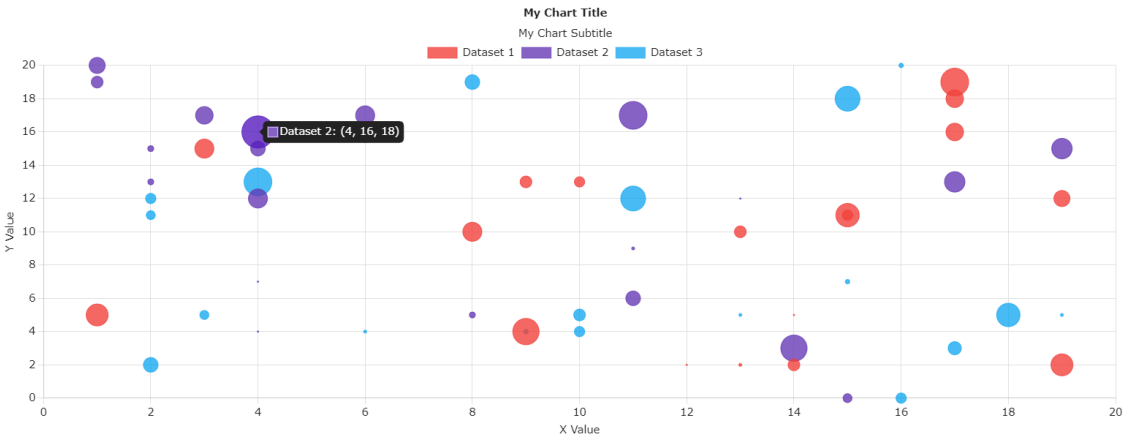
There is a new example called **JS Charts** in the **Samples** section of the **Hub** in the Studio Browser demonstrating all the types of chart available (note there is a New option to display the new examples only). The following image is a **Labelled Bar** chart in the example app:



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).



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.



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.

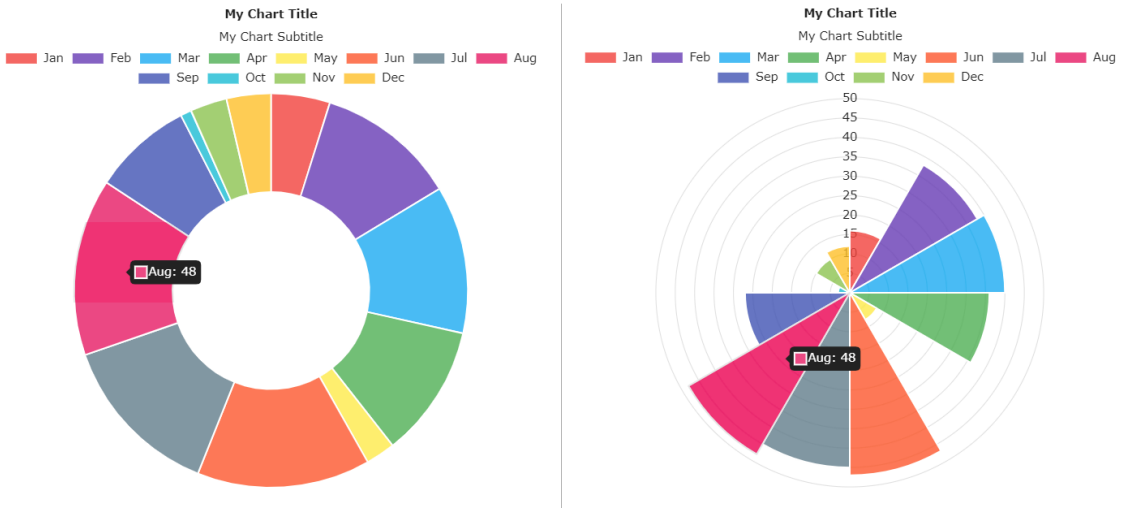


Chart Data

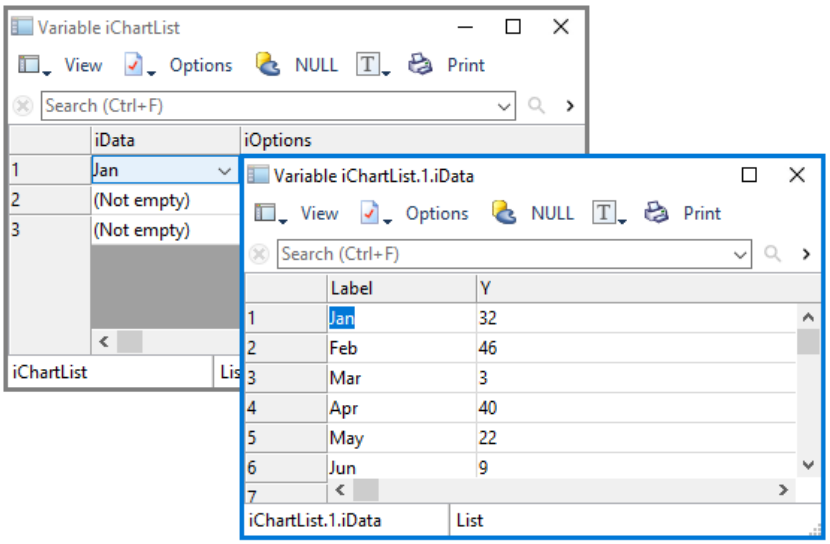
As with other chart types in Omnis, the JS Chart control gets its data from an Omnis *list variable,* and the structure or contents of the list needs to match the type of chart you wish to draw. The chart list should contain 2 columns, with *each row in the list representing a dataset:* Column 1 is the 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.

Chart List Variable		
List line	**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	…	…

The ***Data list*** for the chart (in column 1) will vary depending on the chart type as follows:

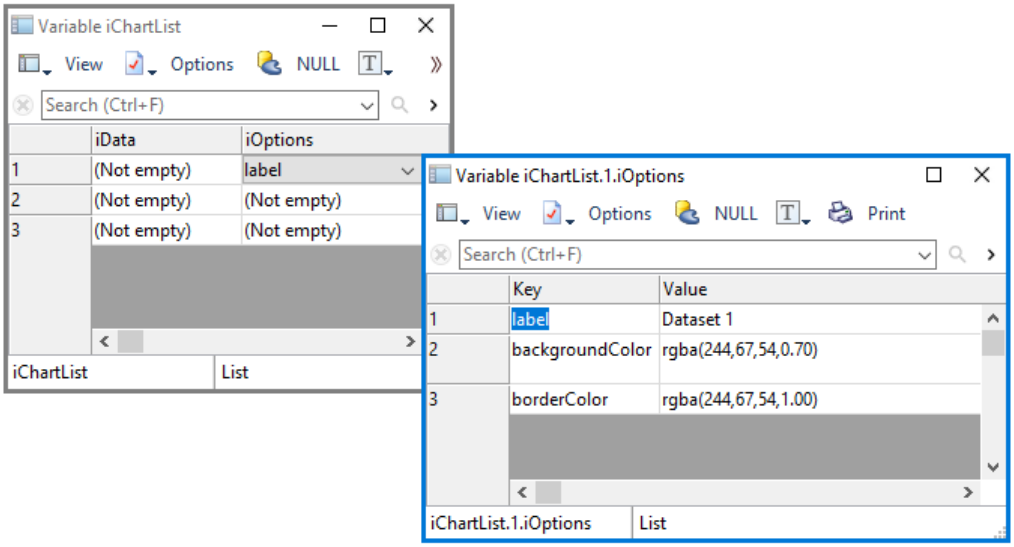
- ❑ The data list variable for *Labelled* and *Area* chart types (e.g. Bar and Pie) requires 2 columns, usually with a Label and a Value:
  Column 1, X axis: Label type data, such as months, exam grades, etc.
  Column 2, Y axis: Value, such as average temperature, number of students, etc.
- ❑ The data 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:

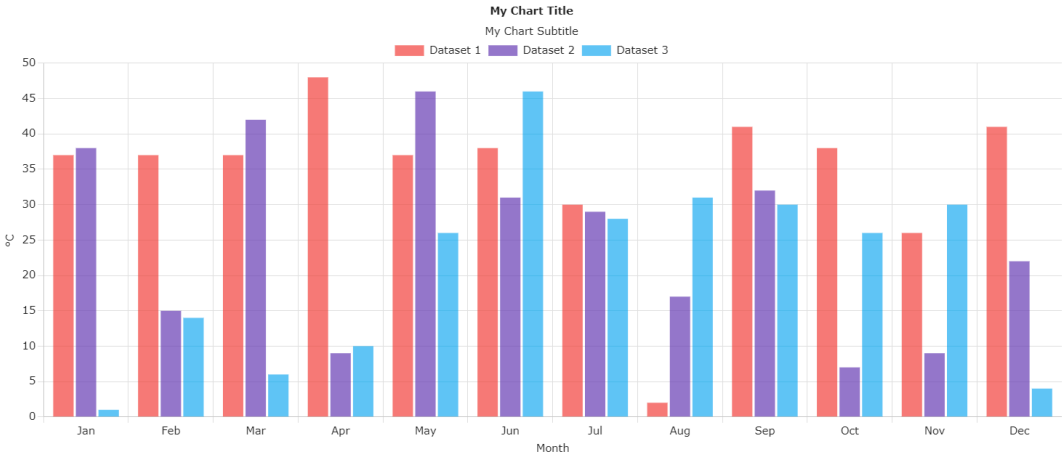


The ***Options list*** in Column 2 of the main chart list must be a list of 2 columns containing key-value pairs of Options to apply to that dataset, which are generally display options (colors/rounding on bars/etc).

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.
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 JS 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	The position of the title, subtitle, and legend, a constant: kJSChartElementPositionTop,

$subtitleposition $legendposition	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 JS Chart control sends the **evClick** and **evLegendClick** events with the following event parameters:

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

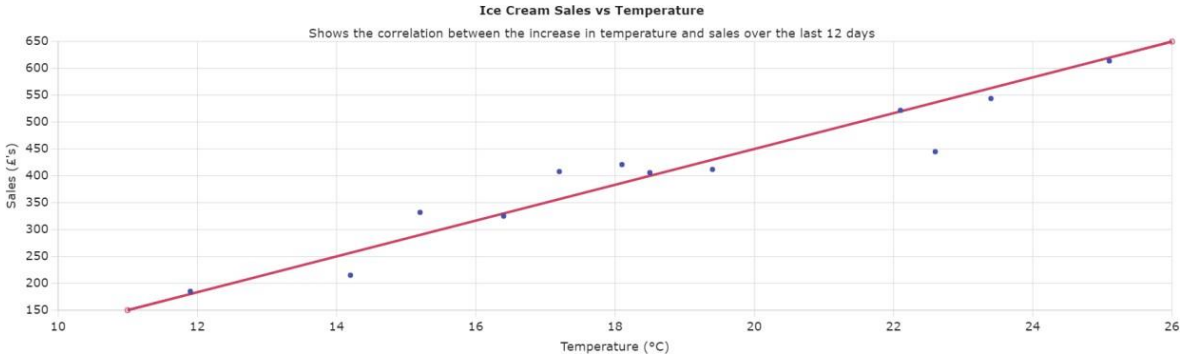
If $legendclickhidesdata is true (the default), when you click on an item in the legend it is toggled on/off and the dataset in the chart is hidden or shown; its state is reported in the pHidden parameter for evLegendClick.

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)
```

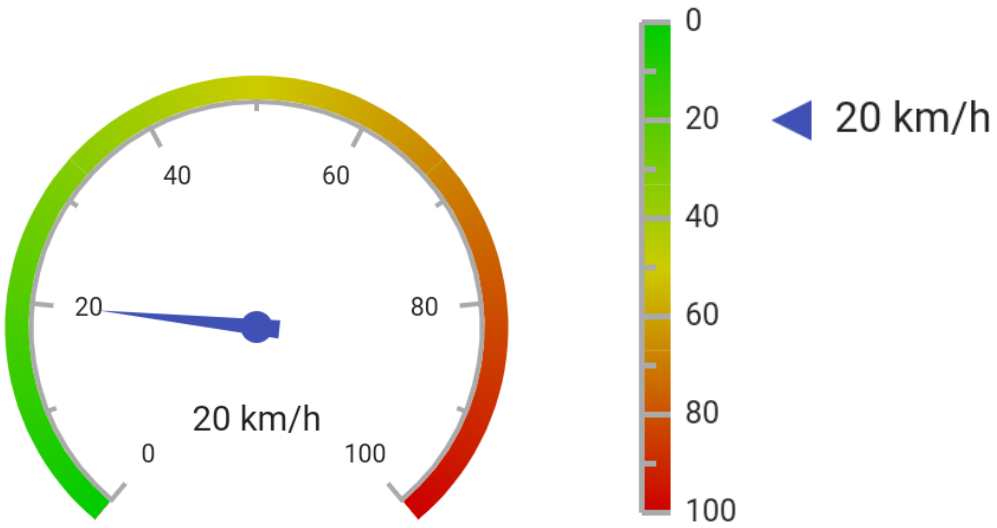


Note how the second dataset, used to portray a line of best fit, is calculated manually, i.e. there is no function to calculate an actual line of best fit.

JS Gauge

The **Gauge** is a new JS component that 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 a new example app called **JS Gauge** in the **Samples** section of the **Hub** in the Studio Browser (note there is a New option to display the new examples only) 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.

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

Property	Description
$opposeaxis	If true, the position of the axis is opposed, e.g. scale on circular gauge is shown on the outside
$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

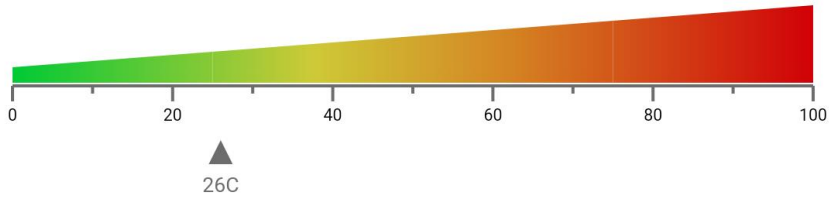
The **evValueChange** event is triggered when the value is changed by the user clicking on the gauge area or dragging the pointer (needle or marker). The **pNewValue** parameter holds the new value.

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:

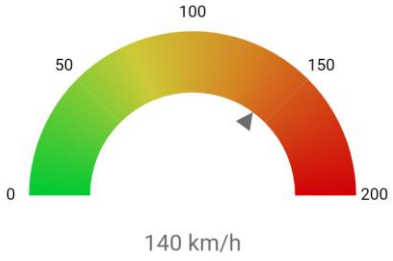
- On the General tab in the Property Manger, **$scalevaluestart** and **$scalevalueend** are set to 1 and 100, respectively.
- On the Appearance tab, **$gaugetype** is set to kJSGaugeTypeHorizontal, **$displayvalue** is set to %dC, **$padding** is set to 100-10 (100 at the top to display the customized range, 10 for each side), **$rangeoffset** is -2 (which provides a gap between the range and scale baseline), and **$tickintervalmajor** is set to 20.

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 (209,1,8)
$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.



JS Camera

The **Camera** control is a new JS component that allows the end user to capture images or scan QR codes and barcodes. 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.

There is a new example app called **JS Camera** under the **Samples** section of the **Hub** in the **Studio Browser** (note there is a New option to display the new examples only).

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 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
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.

JS Floating Action Button

The **Floating Action Button (FAB)** is a new JS component that 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.

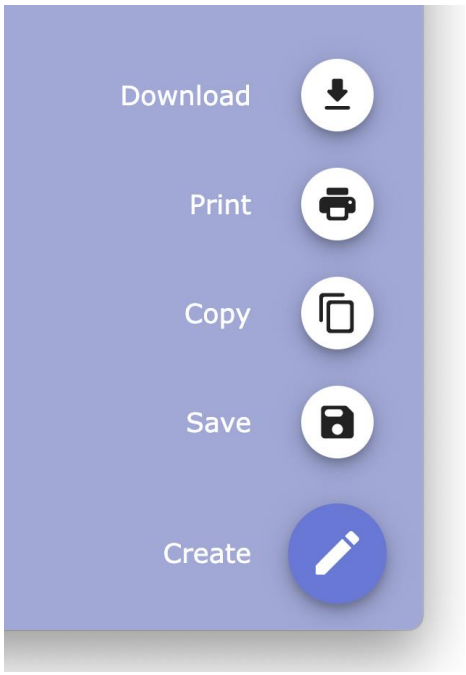
The FAB is displayed as a circular button containing a '+' icon prompting the end user to tap it or hover over it; the default icon can be replaced by setting $iconid of the button control. In its expanded state, the actions in the list appear to "float" on top of the other content in the form.

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 a new example app 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 (note there is a New option to display the new examples only). 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")
```

Properties

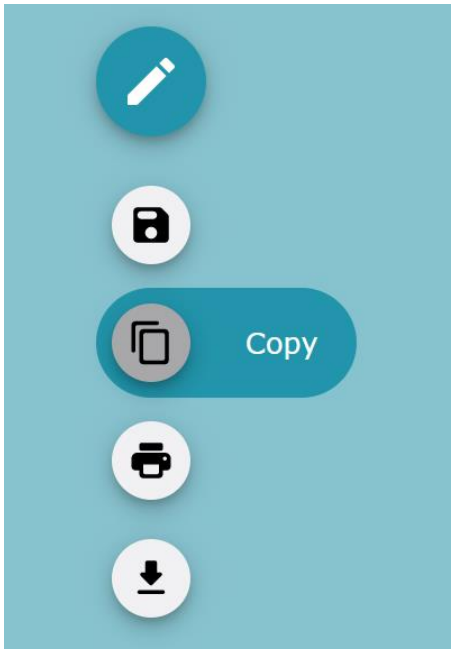
The Floating Action Button has the following properties.

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)
$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



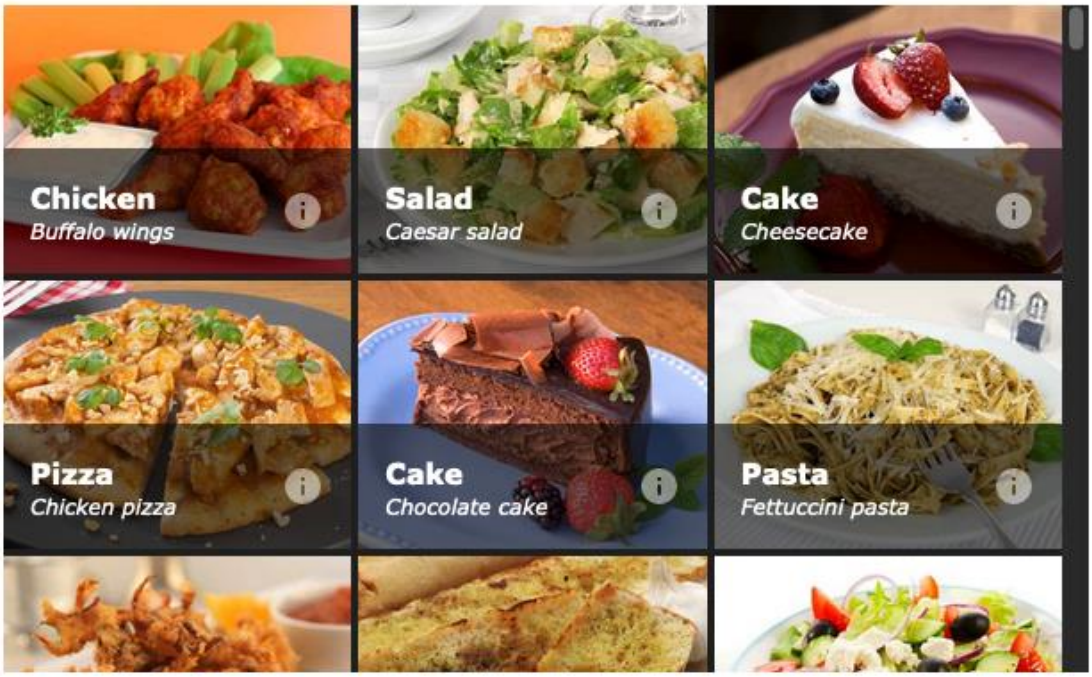
Events

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.

JS Tile Grid

The **Tile Grid** is a new JS component that 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 a new example app called **JS Tile Grid** in the **Samples** section of the **Hub** in the Studio Browser (note there is a New option to display the new examples only), 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
$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
$text1align	The text alignment for the primary text field in the tiles

Property	Description
$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.

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", kJSThemeColorPrimary)
Do iData.$add(
   "images/webshop/Caesar_Salad.jpg", "Salad", "Caesar salad", iconurl("info"),
   "Info", kJSThemeColorPrimary)
Do iData.$add(
   "images/webshop/Cheesecake.jpg", "Cake", "Cheesecake", iconurl("info"),
   "Info", kJSThemeColorPrimary)
# etc
```

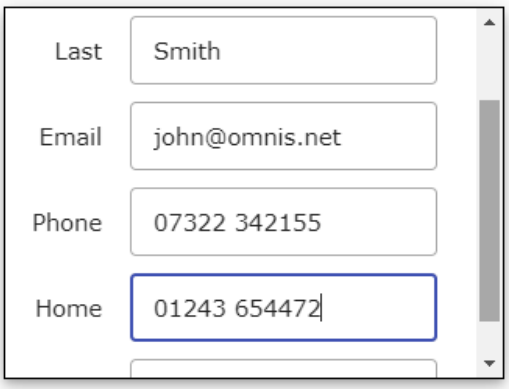
In addition to using bitmap images (JPG or PNG), you can add an SVG image from an icon set to the background of a tile. In this case, you can use the iconurl() function to reference the SVG image.

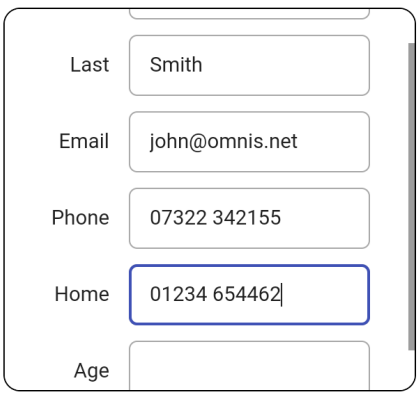
Events

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.

JS Scroll Box

The **Scroll Box** is a new JS component that 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 (it is almost identical to the existing window class Scroll box).





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 above 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 right).

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 use the $makegroupbox() method to convert a Scroll box into a Group box, 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. (ST/JS/2999)

Setting the $label property for a scrollbox 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.

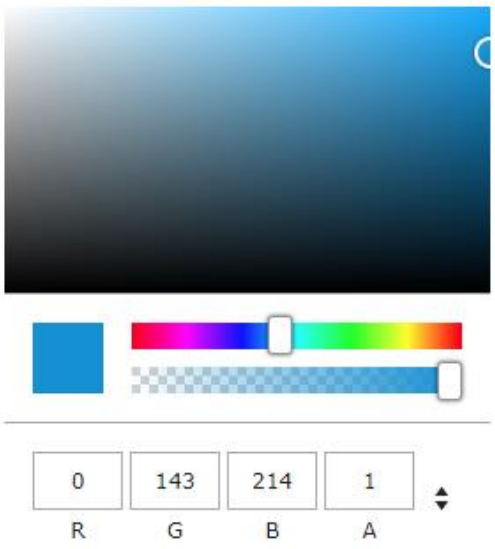
As with other controls with the $label property, you can double-click on the label text in design mode to edit the text.

JS Color Picker

The **Color Picker** is a new JS component that 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.

There is a new example app 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 (note there is a New option to display the new examples only).

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.



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.

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.

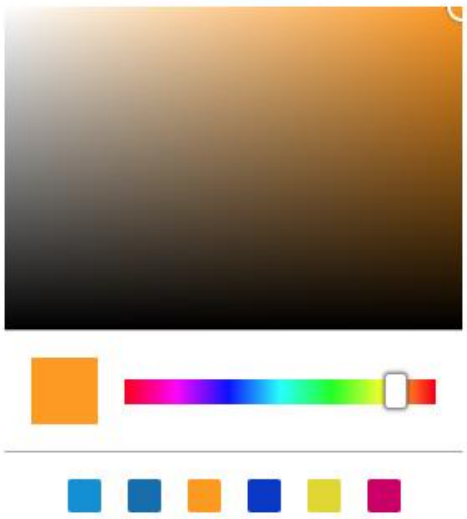
```
On evColorChange
   Calculate iColorChange as pColor
```

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.



JS Side Panels

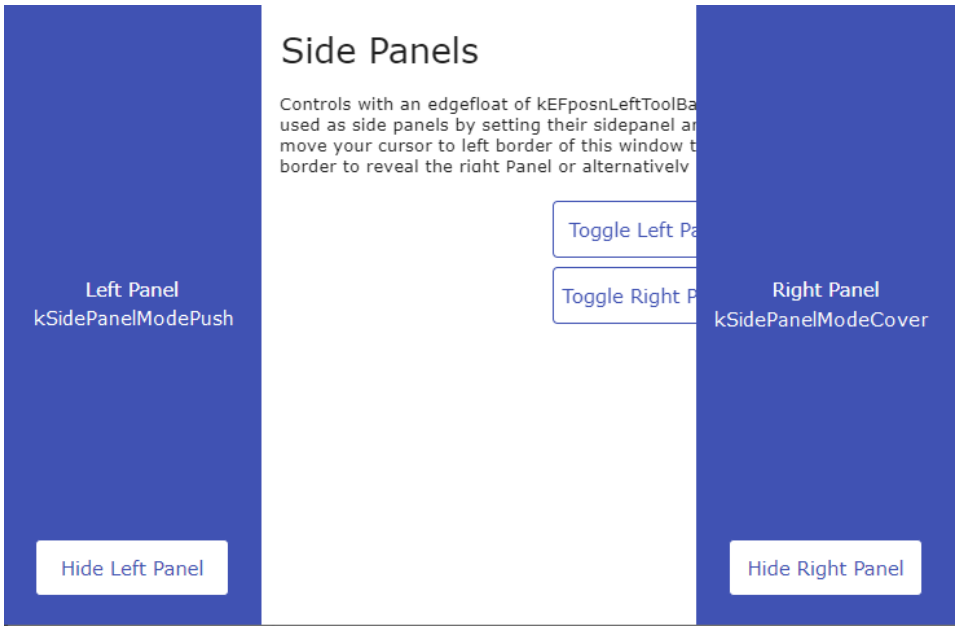
Side Panels were introduced in Studio 10.2 for Window classes, but they are now available for JavaScript Remote forms (see differences at the end of this section).

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 a new example app called **JS Side Panels** under the **Samples** option in the **Hub** in the Studio Browser (note there is a New option to display the new examples only), 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)
- **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.

Existing users should note: Side panels on the JS client work similarly to the fat client, however you should note the following differences from fat client side panels:

- In responsive JS forms with multiple layout breakpoints, a component can be a side panel in one breakpoint and not in another. Therefore, the $sidepanel property can be set independently to $edgefloat, but will only behave as a side panel in breakpoints where $edgefloat of the object is kEFposnLeftToolbar or kEFposnRightToolbar (note in the fat client $edgefloat has to be set before the $sidepanel property is enabled and can be set)
- A side panel's state (visible or hidden) will persist between breakpoints.
- In responsive JS forms, push mode causes the container (which can be the form, a subform or a Paged pane) to resize responsively. Once the container reaches its minimum width for the current breakpoint, it becomes scrollable.

JS Data Grid

There have been a number of enhancements added to the **JS Data Grid** control, including the addition of horizontal padding for grid cells, and more control over which grid lines are displayed when setting the $gridlinesvisible property.

Enter Key Behavior

A new property **$entertodoubleclick** has been added to the Data grid (as well as the JS List, JS Tree list, and JS Date picker) to force the **Enter** key to be *interpreted as a double-click.* (ST/JS/2946)

When $entertodoubleclick is true, the double-click event is sent when the focus is on the list and the Enter key is pressed, allowing more control from the keyboard for the lists and grid controls.

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.

Horizontal Padding

The **$horzpadding** and **$columnhorzpadding** properties have been added to the JS Data Grid to allow you to set the horizontal padding for all the cells in the grid, or for individual user-defined columns. (ST/JS/2405)

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 now 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. (ST/JS/2882)

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 +
   kJSDataGridVisibleGridLinesColumnHeaderHorz
```

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.

Column Headers

The **$columnheadersbold** property has been added to the JS Data grid to allow you to display the headers in bold. (ST/JS/2684)

evCellChanged

The evCellChanged event is now triggered when the end user tabs out of the last line of an extendable grid to create a new line, that is, when the focus is moving to the new line. (ST/JS/2991)

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.

Formatting cells

The $formatcell() method is now fired when the selection state in a data grid is changed, plus a new parameter pSelected has been added. (ST/JS/3101)

The $formatcell() client method is now 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 Properties in Field Styles

Field styles can no longer have multi-value column properties, such as where you can set a different value for each column in a data grid. (ST/EC/1738)

The Property Manager and the interface for the #STYLES system class no longer allow multi-value column properties to be assigned as custom styles. In this case, the Property Manager context menu item "Add To Style As Custom Property..." will be disabled, and attempting to drag to the #STYLES window will now fail.

JS Edit Field

Dynamic Labels

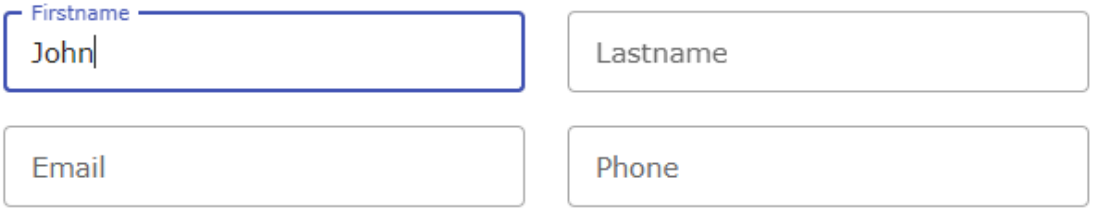
You can now add a dynamic or "floating" label to **Edit fields, Droplists,** or the editable part of **Combo Boxes.** This enhancement means you can choose not to add separate text labels for the fields in a form, and rely on these new field-based dynamic labels. (ST/JS/2838).

A number of properties have been added to the JS Edit, Droplist, and Combo box 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
$labelposition	The position of the label when not shown as the content tip inside the field, a constant: kJSLabelPosBorder (the default), kJSLabelPosAbove, kJSLabelPosLeft

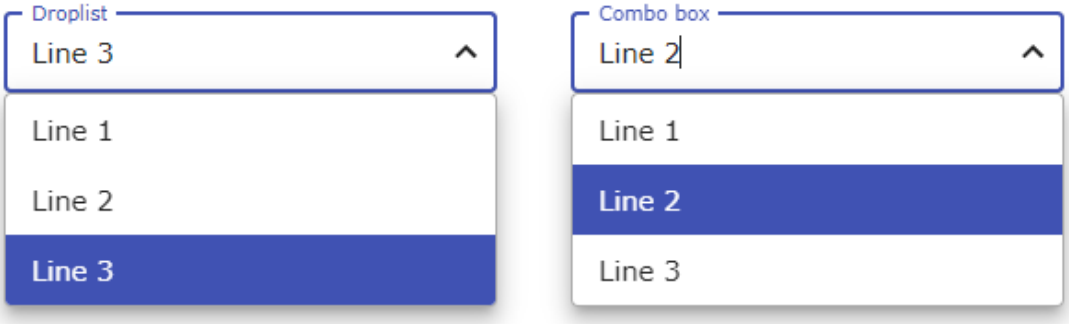
To enable the dynamic label, you need to add the label text to the **$label** property for the control. Once you have added text to the $label property, you can double-click on the label to edit the label text (pressing Return confirms an update). By default, the text label is inset into the top border of the control ($labelposition = kJSLabelPosBorder), unless $labeliscontenttip is true, but $labelposition can be changed to above or left of the control.

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.



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).

You can set the $label property for **Droplists** and **Combo boxes** in the same way; by default the text in $label is displayed inset into the top border of the control.



Content tip text color

The **$contenttiptextcolor** property has been added to JS **Edit fields, Droplists,** and **Combo boxes** to allow you to set the text color for content tips. (ST/JS/3061)

The $contenttiptextcolor property is the text color used for the content tip text, or the label when displayed as a content tip, for an edit field, droplist, or the editable part of a combo box. The new color property applies when using the $::contenttip property or $labeliscontenttip in conjunction with $label.

Date Picker

The **$datepickeroptions** property has been added to Entry fields and Data grids to allow you to customize display options in a calendar style popup date picker, plus the ability to display the isoweek number has been added. (ST/JS/2974)

$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. There are new constants for $datepickeroptions: 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.

JS Button

Text Position

A number of layout properties have been added to the **JS Button** control to give you greater control over the positioning of the text and icon on the button; this enhancement also applies to the **Trans** button and **Split button** components. (ST/JS/3013)

The new layout properties are:

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.

In addition, $horzpadding has been added to the **Trans** button only (the other buttons have it already) to allow you to set the left and right padding.

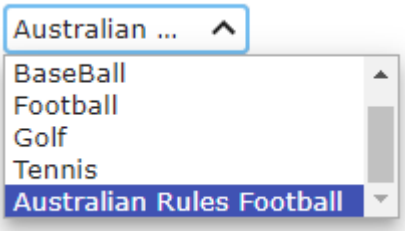
JS Droplist & Combo Box

Droplist Style

The **$droplistst yle** property has been added to the **Droplist** & **Combo box** controls. (ST/JS/2790)

The $droplistst yle property allows you to apply a rounded style to the list part of a Droplist or Combo box control. The style of the droplist is a kJSDropListStyle... constant:

- **kJSDropListStyleDefault**
  The default droplist style (see below left).
- **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 (see below right).



$droplistst yle = kJSDropListStyleDefault (shown on Windows)

$droplistst yle = kJSDropListStyleRounded (shown on macOS)

JS Date Picker

Disabling Dates

The calendar style **Date Picker** now allows you to disable specific dates and to set the start and end dates (minimum and maximum). (ST/JS/3104)

The date picker control has the following properties to allow you to disable dates:

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
$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).

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) a new method, $getdisableddates(), has been added to the controls that support the popup date picker. This defaults to client-executed, however, 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 now display the week number in the calendar view of the Date Picker control by enabling the new property $showweeknumber and setting the associated color properties. (ST/JS/2924)

When set to kTrue, the new **$showweeknumber** property displays the iso week number on the left side of the calendar style date picker (when $datestyle is set to kJSDatePickerStyleCalendar).

The new property **$weeknumbertextcolor** specifies the text color of the week numbers, and **$weeknumbercolor** controls the background color of the week number area.

Calendar View Change Event

A new evCalendarViewChanged event has been added to the Date Picker component. (ST/JS/2925)

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)

JS File

You are now able to drop files onto the **File control** to upload them. (ST/JS/2689 and ST/JS/3063)

There is a new example app called **JS File Upload/Download** under the **Samples** section of the **Hub** in the **Studio Browser** (note there is a New option to display the new examples only).

The input area in the File control is now larger, and allows you to drag and drop files onto it for uploading. A generic progress spinner has been added, 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.

The file control can now be used as an upload control on a form when the new property **$showinline** is 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 **$hideuploadprogress** is true, the upload progress area is hidden. The uploading spinner has been added to the upload area to simplify the control if both upload progress areas are switched off. Plus **$hideuploadprogressbatch** has been added to hide the total batch upload progress area.

The **$uploadprogresstextcolor** property is the color for the text in the progress elements.

The **$clearsfileselection** property clears the last uploaded file. When set to true, the current file selection is cleared automatically after an upload has completed.

In addition to the upload enhancements, the new **$maincolor** property is the main color used throughout the control, including the upload area, upload button, progress bars, completion indicators, and upload spinner.

JS Slider

The **$reversescale** property has been added to the **JS Slider** control to allow you to reverse the scale on the slider. (ST/JS/2846)

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.

JS Toolbar

The **$clippopuptocontainer** property has been added to the **JS Toolbar** component. (ST/JS/2956)

If true (the default), the Side menu and Overflow menu are clipped to the toolbar's container. If false, they can extend outside its bounds.

JS Nav Bar

The **evWillPop** event has been added to the **JS Nav bar** control. (ST/JS/3048)

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)
```

JS Map

SVG marker color

You can now add a color to a custom SVG map marker when specifying the list of markers in a JS Map. (ST/JS/3027)

When an SVG icon is used in the map markers list, you can now 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.

Border Radius

The **$borderradius** property has been added to the **JS Map** control to allow you to set a border radius. (ST/JS/3023)

JS Native List

Menu Accessory

A new 'menu' accessory type has been added to the JS Native List. (ST/JS/2837 & ST/JS/3097)

The new **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.

Vertical Scroll

The $vscroll property has been added to the JS Native List component which allows you to dynamically scroll the list to the specified line. (ST/JS/2962)

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.

JS Picture

The **$tintcolor** property has been added to the **Picture** control to allow you to apply a tint color to themed SVG icons. (ST/JS/2964)

The new property will only apply the tint color to SVG images that have been themed using the **JS Themer** tool (available in the **Tools>>Add ons** menu), so for all other image formats the tint color is ignored.

JS Rich Text Editor

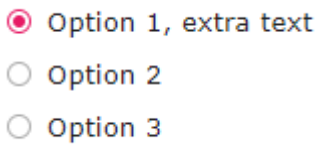
The $gethtmlwithstyles() method has been added to the JS Rich Text Editor control to provide a better representation of the html in the control. (ST/JS/3093)

The **$gethtmlwithstyles()** client method returns the HTML from the Rich Text Editor control and applies some inline styles to the elements, which when viewed externally, such as in a web browser, should better 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.

JS Radio Button Group

It is now possible to include a comma in the text for a button in a JS Radio Button Group (also applies to window class radio buttons). (ST/JS/3202 & ST/WO/2717)

JS Radio button groups now support a second comma to escape a comma when specifying $text for a number of buttons. 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:



Icon Badges

You can now 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.) (ST/IF/351)

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:

- **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 color 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:

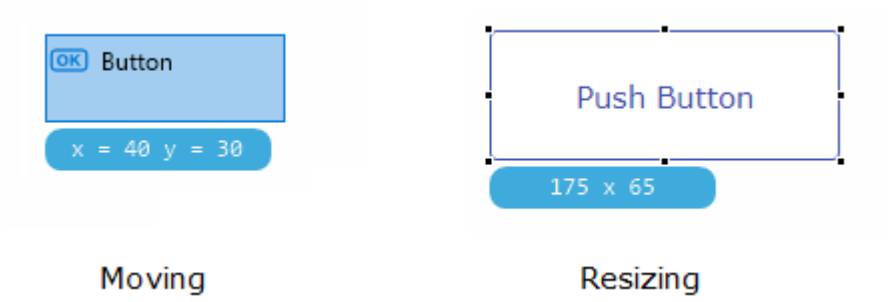
```
Do $cinst.$objs.tabpaneorstrip.$settabinfo( 1, '', iconidwithbadge(
   'tablet_mac', 1 ) )
```

Position Assistance

The following new positioning and sizing assistance enhancements have been added to help you add or move objects on a remote form class (or window class design window).

Position and Size coordinates

The **Position Assistance** capability now provides *position* and *size* information when you *move* or *resize* an object, or group of objects, in a remote form class (or window class). In addition, *position* information is provided when you drag an object from the Component Store and drop it onto a remote form. (ST/HI/1914)



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 design screen.

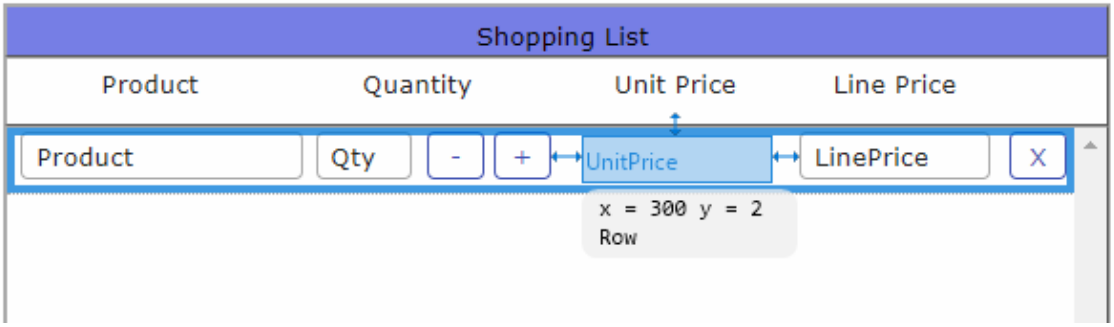
The current ***Size*** of an object is shown (width x height) when you **resize** it (see above right). When more than one object is selected, the position or size corresponds to the area of the whole group of selected objects.

There is a new item **positionAssistantShowsPositionOrSize** in the 'ide' section of the config.json file that allows you to enable or disable this feature (the default is true, so the position or size is shown).

Position for dropping objects

Information about the drop destination is now 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). (ST/WO/2686)

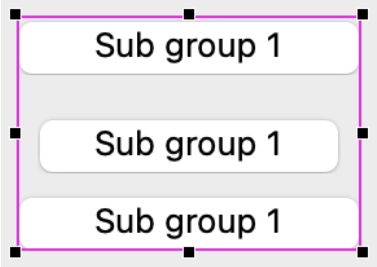
For example, as you drop a field into a Complex grid section, the position assistance will display the section type, such as 'Header', 'Horizontal header', or 'Row', as shown below.



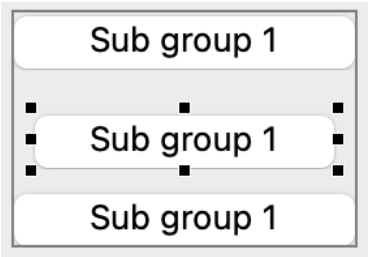
Group Selection & Object Properties

The following enhancements have been added to make it easier to examine the properties of an object on a remote form *when it is within a group of selected objects,* or part of a linked group or container field. (The enhancements also apply to selecting objects in Window and Report classes).

When you select a group of objects, Omnis now shows a colored line around the group and *a single set of selection handles* for the group. 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.



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.

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 new 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

SVG Icons

Due to support for themed SVG Icons being added to window class controls, some changes have been made to the SVG Themer Tool and the material icon set (available in previous versions of Omnis Studio 10.2).

SVG Themer tool

The **SVG Themer** tool is now available on the **Tools >> Add-ons** menu, and is not found in the Web Client Tools submenu, as in previous versions.

Material Icon set

The **'material' icon set** folder has been moved into the 'iconsets' folder in the Studio tree; it was in the 'html/icons' folder in previous versions of Omnis Studio 10.x. If you use any icons in the 'material' icon set in your web or mobile apps, you need to copy the material icon set to the 'html/icons' folder when deploying to a web server.

SVG Icon size

When choosing an SVG icon in the Property Manager you can now specify the icon size, in addition to choosing the existing standard sizes. (ST/HI/2005)

When specifying **$iconid** in the Property Manager, the id edit field now 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.

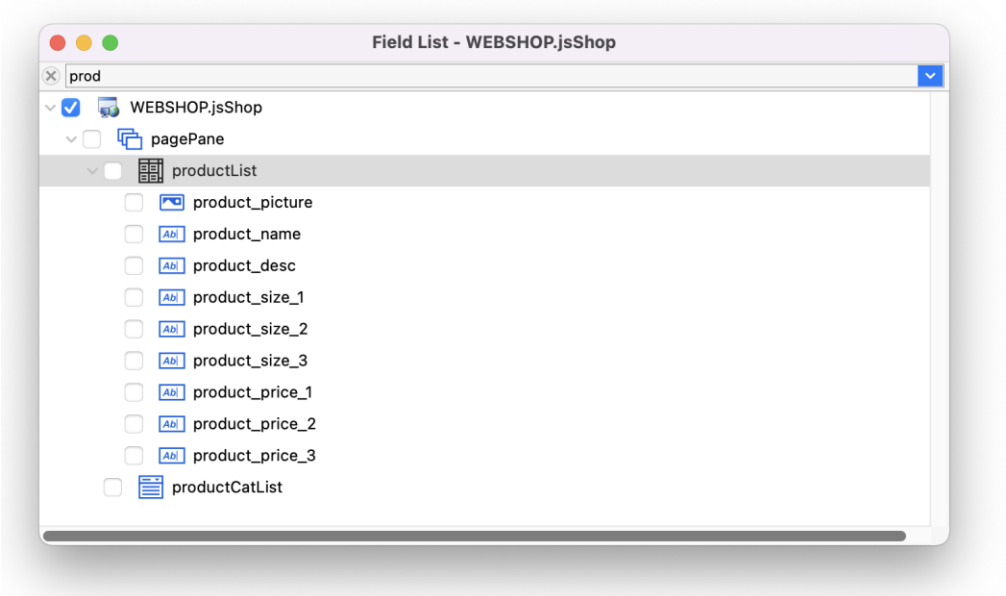
Field List

The following enhancements apply to the **Field List** for Remote form classes, as well as Window and Report classes.

Object Search

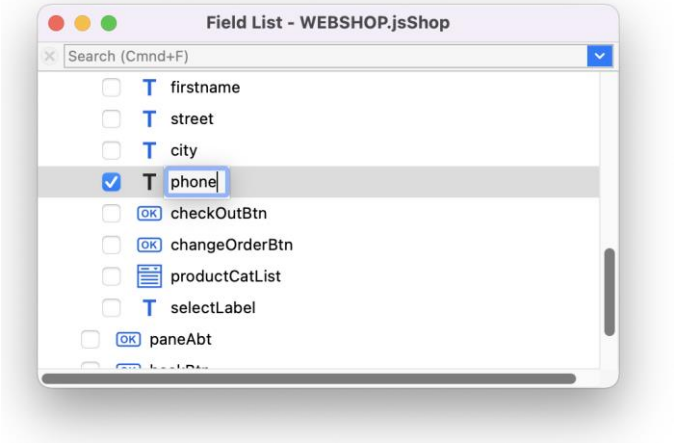
A **Search** box has been added to the Field list to allow you to locate controls or background objects in a remote form. (ST/WC/576)

The search 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

You can now rename a JavaScript 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.



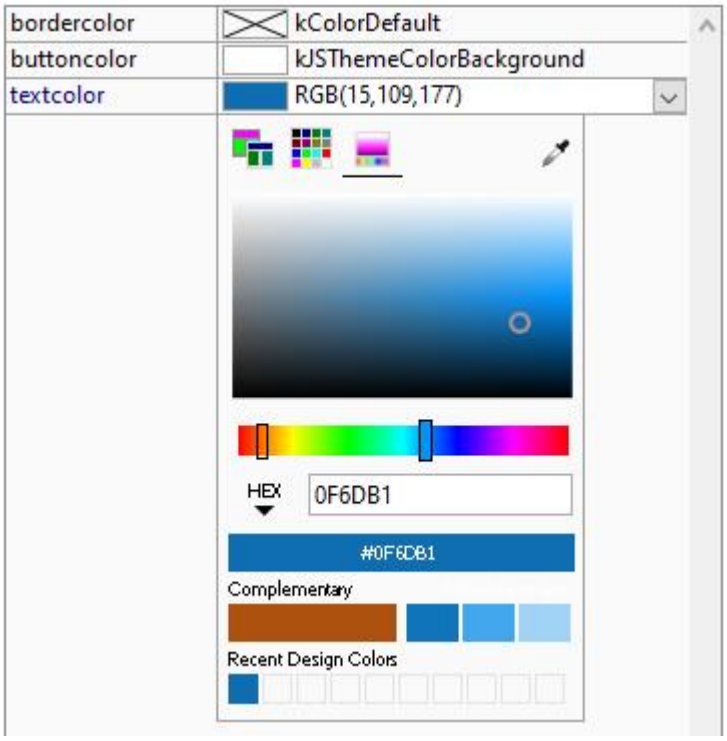
When you rename an object on a remote form (using the Property Manager or Field List), Omnis searches for any properties using the old name, and replaces them with the new name, including properties such as $arialabelledby and $linkedobject.

Opening the Field List

You can now open the Field List using the F7 key when editing a Remote Form. The setting is stored in the 'ide' section of the keys.json file. (ST/HE/1498)

Color Palette

The color palette has been enhanced by adding a new tab to allow you to select colors for components and objects on remote forms (as well as window and report classes). (ST/HE/1820)



The updated color palette allows you to select a color from a color picker, or to enter a color number in RGB, HSL, or HEX format. The new color picker also shows a range of

tints of the selected color, as well as its complimentary color, which may be useful for selecting a set of colors for a remote form design. The recent selected colors are shown at the bottom.

Background Images

You can now drop an image from the system onto a Remote Form to create a background image. (ST/JS/2851)

You can drop an image from your system / desktop on to a Remote Form to create a JS Background component of kJSBackImage type with the $imagepath property 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.

In addition, the **$keepaspectratio** property has been added to the JS Background component, so when it is a background image ($shape is set to kJSBackImage) the image will keep its aspect ratio; it defaults to kTrue. (ST/JS/3058)

When setting the path to a background image in the JS Background component, the $imagepath property now has a prompt button to allow you to select an image. (ST/JS/2868)

Inactive Appearance

The **$defaultinactiveappearance** property has been added to the majority of the JavaScript components to give you more control over the inactive appearance of controls. (ST/JS/2739)

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.

Edge Float

Some new edge-float constants have been added to the $edgefloat property for JavaScript components allowing you more control over how objects will float or resize in a remote form (the same constants have been added to window controls). (ST/JS/2669, ST/WO/2458, ST/WO/2710)

- **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)

Fonts and Semi-bold

The **Roboto Flex** font has been added and is now the default font for all JS components, including Entry fields and labels *in new libraries.* (ST/JS/3230)

Roboto Flex is a 'variable font' and includes many font weights and styles, including semi-bold and extra-bold. 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

In addition, the font named 'system-ui' has been added to JS components which 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.

Studio 11 now supports **Semi-bold** and **Extra-bold** font styles, if the font supports it. The constant kSemiBold allows semi-bold to be set for fields and labels. Using both kBold and kSemiBold causes an extra bold font style to be used.

Tab Order

Remove from Tab Order

The **$removefromtaborder** property has been added to all JavaScript components, except for any components that can't normally be tabbed to. (ST/JS/2751)

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.

Next Tab Object

You can now specify the row when setting the $nexttabobject property to a complex grid child. (ST/JS/3226)

Subform Events

The **evSubformLoaded** event has been added to the Subform control. (ST/JS/2781)

The new 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.

Subform Promise

You can now 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. (ST/JS/3231)

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.

Rounded Borders

The **$borderradius** property has been added to a number of JavaScript components to allow you to apply a more consistent style for all the JS controls in a remote form. (ST/JS/2640 and ST/JS/2691)

A single value sets the radius for all the corners of an object, or to specify a different radius for each corner you can use the syntax "n-n-n-n" which follows the same rules as CSS 3 rounded border syntax, i.e. top-left, top-right, bottom-right, bottom-left.

The $borderradius property has been added to the following JavaScript controls: Tree list, Toolbar, Progress Bar, Nav Menu, Navbar, Native List, Hyperlink, and Complex Grid, plus the new Scroll Box.

Numeric Object Names

The **allowNumericObjectNames** configuration item has been added the 'ide' section of config.json. When false (the default), the Property Manager does not allow all numeric names to be assigned to $name. (ST/PC/575)

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 "+-.".

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.

ARIA Properties

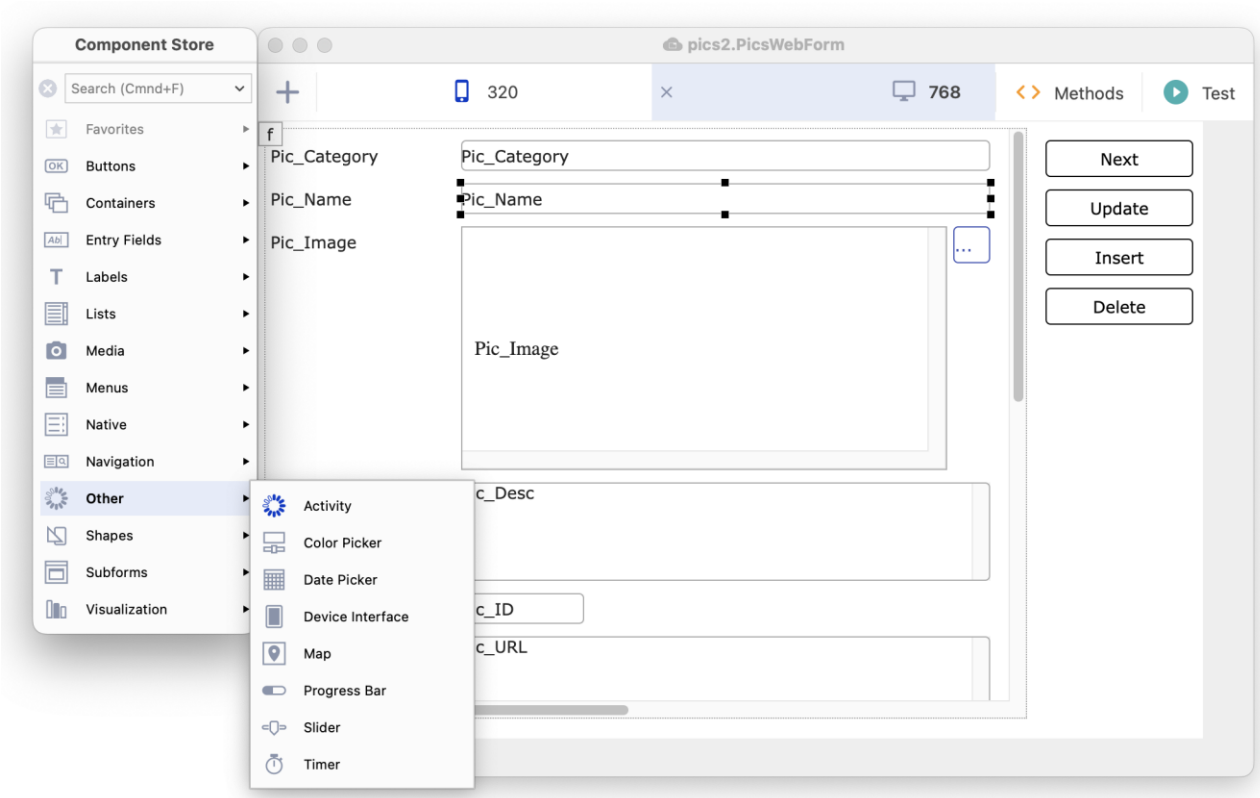
The ARIA properties **$arialabelledby** and **$ariadescribedby** now take a comma separated list of controls, rather than using a space as a separator, as in previous versions; this is to accommodate control names that contain spaces. (ST/JS/2596)

When converting libraries from 10.2 or earlier, or importing a library from JSON, spaces in $arialabelledby and $ariadescribedby will be replaced with commas.

JavaScript Remote Forms

Remote Form Editor

The Design bar in the remote form editor now contains a **Methods** button to open the Method Editor for the form, and a **Test** button to open the form in a web browser (this is the same as the existing Test Form or Ctrl-T option). Together with the new Component Store (seen here on the left) and enhancements in the Property Manager, creating Remote forms is now quicker and easier.

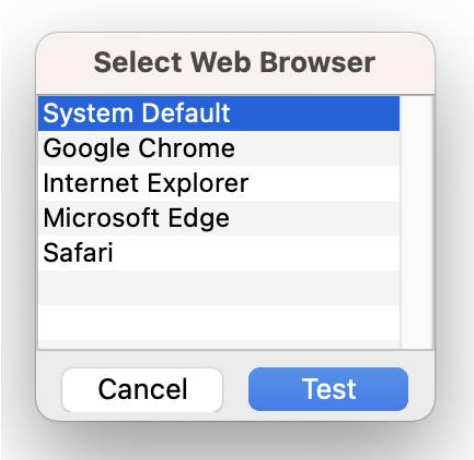


A similar Design bar has been added to the Window class editor to provide quick access to the window's **Methods** and to **Test** the window.

Testing Remote Forms

When testing a JavaScript Remote form in design mode, you can now select the web browser in which to test the remote form. In previous versions, the default system browser was used automatically to test a form (e.g. when using Ctrl/Cmnd-T). (ST/HI/1905)

There is a new menu item on the Remote form design Context menu, **Select Browser And Test Form...** (or you can use the shortcut Shift+Ctrl/Cmnd+T), that 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.



The new Select Web Browser dialog is 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.

Subform Palettes

You can now popup a subform in a palette style window next to a specified control. Such a *Subform Palette* could contain a subform to 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 two new client commands, or a subform palette can be closed by clicking outside the subform. (ST/JS/3156)

The new client commands ($clientcommand) **subformpaletteshow** and **subformpaletteclose,** have been added to support subform palettes.

There is a new example app called **JS Subform Dialogs** in the **Samples** section of the **Hub** in the Studio Browser (note there is a New option to display the new examples only), showing how to use the new client commands and the different settings for palette subforms. (Note the new example app also shows how to display Subform Dialogs using the **subformdialogshow** client command, which was available in previous versions.)

subformpaletteshow

The **subformpaletteshow** command shows a remote form as a subform palette:

```
Do $cinst.$clientcommand("subformpaletteshow",row)
```

Where row variable is row(cClass, cParams, cControl, iWidth, iHeight [,iPositionFlags] [,bShowOverlay] [,cTitle] [,bPreventClose])

The row variable parameters are as follows:

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) a combination of up to 2 kSFSPalette… constants to specify the position (defaults to kSFSPalettePosFlagRight+kSFSPaletteAlignFlagCenter). See further description of flags below.

[**bShowOverlay**]: (Boolean) if kTrue, shows the form overlay while the palette is open (defaults to kFalse).

[**cTitle**]**:** (Character) 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) 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",
   220,120,
   iPalettePosValue+iPaletteAlignValue,
```

```
kFalse,
"MyTitle"))
```

subformpaletteclose

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).

Event Specific Client Methods

The **$eventclient()** method has been added to support client-executed methods 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. (ST/JS/2842)

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

The **Show Built-in Class Methods** option in the method editor **View** menu, has been renamed to **Show Built-in Methods.** This option defaults to on, and when enabled, the method editor tree list shows all possible *Built-in class* and *Control methods* that can be overridden, including $event and $eventclient (in previous versions 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 now have a tooltip that is displayed when the **Show Method Content Tips** option in the View menu is enabled.

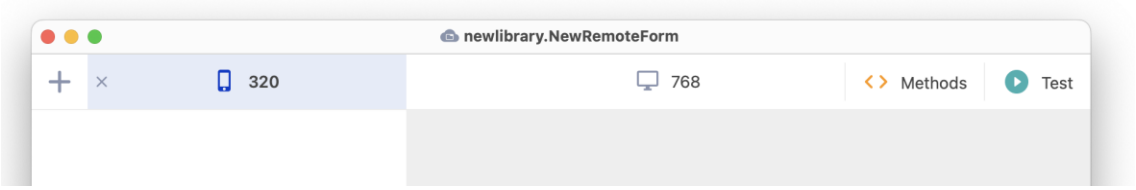
Layout Breakpoints

Minimum number of Breakpoints

In previous versions, the *minimum number* of layout breakpoints for a responsive Remote form was *two,* but that restriction has been removed, meaning that a remote form can now have only one breakpoint, if required. (ST/CE/153)

With this enhancement, you could create a layout for all devices using a single layout, by specifying only one breakpoint, and cater to all possible display sizes or devices using the *floating edge properties* of the JS controls on your remote form. This may be useful if you create a form that will be re-used as a *subform,* where having more than one breakpoint may not be necessary.

When you create a new Remote form in the Studio Browser it will contain two breakpoints (320 and 768), but you can delete one of these and reset the width of the remaining breakpoint, as required.



To delete a breakpoint, click on the X button or select it and press Ctrl/Cmnd-D. To change the value of a breakpoint, double-click on the breakpoint header and edit the number, or drag and resize the right edge of the breakpoint header.

Component Properties in new Breakpoints

When adding a new layout breakpoint, all breakpoint-specific properties are now copied from the nearest breakpoint – in previous versions, Omnis copied just the size and position coordinates of the components in the existing breakpoint, but none of the other component properties. (ST/JS/2818)

The properties that are now copied from the nearest breakpoint include the following: layout padding for the form, edge float, align, drag border, error text pos, and 'visible in breakpoint' properties are copied for all controls.

Copying Layout Breakpoints

The **Copy Layout from Breakpoint** option can now apply to specific selected controls. (ST/HE/1851)

The Copy Layout from Breakpoint menu option now applies 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'.

When no objects are selected, the Copy layout from breakpoint menu option works as before, copying the layout (positions) of all the objects on the form.

Assigning Properties

There are some new options in the Property Manager to help you assign properties to controls in different layout breakpoints in a responsive remote form. (ST/HE/1842)

The option 'Copy <property> To All Other Layout Breakpoints' has been added to the Property Manager context menu to allow you to copy the property value of a control to other instances of the control on all other breakpoints.

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

This is in addition to the existing option to assign a property value of a control to all layout breakpoints when setting a value in the Property Manager.

Subform Sets

When opening Subform windows within a Subform Set, their sizing and positioning has been improved. (ST/JS/3088)

Subform set windows now 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.

Add Return Method

A new option **Add Return Method** has been added to the **Method** menu and the Method editor tree context menu to add a “<method name>_return” method for a server method (applies to Remote forms only). (ST/DB/1368)

A <method name>_return method is a special client executed method to handle the return value from a server method called <method name>.

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 now reported as an error. (ST/JS/3127)

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.

Remote Menus

Text Alignment

You can now change the text alignment in remote menus. (ST/JS/3007)

The pContextMenu event parameter in evOpenContextMenu events now 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.

Icon Colors

Remote menus now support the $iconcolor and $defaulticoncolor properties to control the color of icons when using themed SVG icons. (ST/JS/3240)

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 themed SVG icons *and the $iconcolor property of the item is kColorDefault*. If $defaulticoncolor is also kColorDefault, then themed icons use the text color.

PDF Printing

PDF path names

The character limit of 255 for the path when creating a PDF has been removed. (ST/EC/1709)

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

You can now set the PDF version and encryption used in the PDF file using the new $setpdfversion method. (ST/RC/1390)

The **$setpdfversion** method can be one 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 now 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. (ST/RC/1329 & ST/EC/1775)

A new method **$setpdfsubset** allows you to set the PDF/A subset type. The method takes a constant, one of the following:

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

PDF Keywords

The PDF Device now support keywords via the $setdocinfo method. (ST/FU/836)

You can now assign keywords to a PDF to be added to the file's metadata. The keywords are set through extra parameters added to $setdocinfo, specified as a string of comma-separated keywords, for example:

```
Do Omnis PDF Device.$setdocinfo(author,title,subject,'keyword1, keyword2,
   keyword3')
```

Libraries and Classes

Restoring Open Libraries & Classes

Omnis now opens any **libraries** that were open at shutdown when it next starts up; this applies to the development version only. This is controlled by the new item **restoreOpenLibsAtStartup** in the "ide" section of the config.json file, that defaults to true.

When the development version of Omnis shuts down successfully, it saves a list of libraries to re-open. The library list saved excludes all libraries in the startup and studio folders, and all private libraries (these libraries will typically re-open anyway). In addition, Omnis will only run the startup task of a library that it re-opens, if the startup task was open when Omnis last shut down successfully. Running the startup task is controlled by the **openStartupTaskWhenRestoringOpenLibrary** item in the "ide"

section of config.json. If this is true (the default), Omnis will run the startup task of each library that had an open startup task when Omnis closed. Set this to false if you do not want the startup tasks of libraries to be run.

In addition to libraries, Omnis now opens any **class editors** that were open at shutdown when it next starts up; this applies to the development version only. This is controlled by the new item **restoreOpenClassEditorsAtStartup** in the “ide” section of the config.json file (the value of this configuration entry is ignored, and treated as false, if the restoreOpenLibsAtStartup entry is false). If true (the default), after completing startup, the development version of Omnis tries to re-open class editors that were open when Omnis last shut down successfully. Note the system table editors are not reopened.

For class editors other than remote form and window editors (which automatically save their last position), the restored editors open in their last screen position, provided that the screen configuration has not changed since Omnis was shut down; if the screen configuration has changed, then the editors open at their default position for the new screen configuration.

The method editor attempts to restore the selection to the method line being edited. The remote form, report and window editors attempt to restore their current selection. These attempts will work unless the class has been changed, that is, by replacing the library (or class) with a modified copy before restarting, e.g. from the VCS.

The restored class editors open behind any user windows opened by either startup libraries or libraries opened due to the restoreOpenLibsAtStartup config.json entry.

Closing All Libraries

The **Close All Libraries** command has been added to the **File** menu in the *Runtime* and *Server* versions of Omnis Studio to enable you to close all libraries in a single command, rather than closing them individually; *note this option is not available in the Development version.* (ST/BR/389)

Open/Close Library Notifications

There is a new task message $openlibschanged that is sent after a library or libraries have been opened or closed. It is sent in a development version of Omnis only. (ST/EM/226)

Class Names

Extra validation when naming an Omnis class has been added to prevent certain characters from being used in class names, including the characters ( ) [ ] . (that is, parenthesis, square brackets and the period/full stop character). (ST/BR/392)

There is a new config.json entry, **extraClassNameValidations** in the ‘defaults’ section of the config.json file; the default is True meaning that Omnis performs extra validations before assigning a name to a class.

This extra validation is recommended, as using the characters it excludes in class names can cause confusion and potentially errors. This property is present to allow code from previous versions to continue working if the additional validations cause any code to fail.

The characters excluded by these validations are any character with a value less than space, a character in the string ".,;:!?)]}([{+-*/|&><=", or a single or double quote character. In addition, the validations do not allow the name to start with the dollar character $.

Library APIs

A new method **$getapiobject** has been added to $root.$modes to return a reference of an object in another library, allowing you to use its methods, even if the library is private. (ST/FU/787)

You can use $root.$modes.$getapiobject("libraryA") from libraryB to call into $getapiobject method of libraryA startup task which must return an object reference. For example, the startup task method $getapiobject of libraryA can do:

```
Quit method $clib.$objects.libAPI.$newref()
```

Where libAPI is an object within the library which implements some methods that you can use. If libraryA does not return an object reference, the returned value to the caller is NULL. The startup task in the called library must be named 'Startup_Task' (the default name) in order for this to work.

Library Internal Names

The internal name of a library is now prevented from being the same as a static function group name, such as FileOps, to avoid any conflict in your code when a function in a static function group is called. (ST/PF/1259)

Omnis now prevents the internal name of a library from being set to the name of a static function group, by appending a digit (or digits) to the internal name that would otherwise be used, in the same way as it does when opening a library which would result in a duplicate internal name. So in the case of FileOps, the internal name of the library would typically be fileops1. However it is best to avoid using a function group name, or any other function or command name, as a library name to avoid any possible conflicts.

Importing Libraries

The $importjson() method now allows you to replace an existing library when importing a library from a JSON export file. (ST/AD/258)

The $importjson() method has a new parameter bReplaceExisting to import and replace an existing library. If bReplaceExisting is true (the default is kFalse), $importjson() closes the library if it is already open, backs it up to the import archives folder, and imports the library to replace the backed up version; if the import fails, $importjson() restores the original library from the archived copy. Omnis keeps the last 10 archived copies.

The archived copies are stored by default in the archives folder in the Omnis data folder. Each library has its own sub-folder in archives, named using the library name. You can override the archive folder by setting the **archivefolder** member of $prefs.$exportimportjsonoptions Omnis preference.

Startup Task

The $external property of the Startup_Task in a new library is now marked as kTrue (in previous versions is was initially set to kFalse). (ST/TC/041)

Library Startup & Conversion

A new parameter has been added to the $add() method for the $libs group to allow the Omnis Runtime to convert a library or to not run the Startup_Task (ST/FU/840)

The iFlags parameter has been added to the $libs.$add() to allow different options to be set when the Omnis Runtime opens a library. The new iFlags parameter is a sum of one or more of the following kLibFlag... constants:

Constant	Description
kLibFlagNone	If specified, has no effect

kLibFlagDoNotOpenStartupTask	If specified, do not open the Startup_Task
kLibFlagEnableConversionByRuntime	The Omnis runtime version will offer to convert the library
kLibFlagConvertWithoutUserPrompts	If specified, and conversion is allowed, Omnis will immediately perform conversion without giving the user any prompts that require a response; note the user cannot cancel the conversion in this case

Any further parameters are passed to the $construct method of the Startup_Task (in the case where the Startup_Task is allowed to run).

Method Editor

The following enhancements have been added to the Method Editor or Code Editor.

Conditional Breakpoints

You can now add conditions or a hit count to Breakpoints. To enable you to add a condition, the **Set Condition...** option has been added to the Breakpoint context menu, the Breakpoints toolbar menu and Breakpoint list context menu. (ST/DB/1177)

The Set Condition... option allows you to enter a calculation that must evaluate to true (non-zero) for the breakpoint to be hit, and/or a number of hits that are to be ignored before the breakpoint is hit. The calculation and/or ignore count is displayed in parentheses in the breakpoint list.

The remote debugger displays the remote debug breakpoint, although it does not include a hit count.

To allow conditional breakpoints to be stored, a new system table called #DEBUGGER has been added to Omnis libraries to save current local debugger code breakpoint locations, which means code breakpoints (and their conditions) are restored when a library is reopened. #DEBUGGER does not appear in the Studio Browser class list, but is included in $clib.$classes.

List Variable Search

A **Search** panel has been added to the List variable window in the debugger to allow you to search the contents of a large list variable while debugging. (ST/DB/1032)

To initiate a search, click into the List variable window, press Return, and then press the Search button, or select a previously saved search in the droplist. There is also a button to navigate to the current line, which sounds the bell if pressed when there is no current line.

Search results appear in a separate panel that allows you to quickly navigate to results. The Line number and Column name for each search result is shown. If the Column name is empty, the Column number is shown.

Find Possible Calls

A new option **Find Possible Calls...** has been added to the Context menu available in the method list in the Method Editor to locate all possible calls to the method. (ST/FR/162)

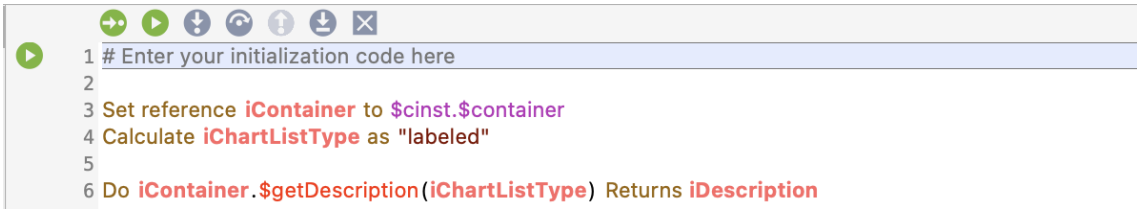
The Find Possible Calls... option attempts to locate all possible calls to the method, from methods in the current library, or a selected set of open libraries. If there is only one open library, it performs the search immediately, displaying a progress bar. If there is more than one open library, the option opens a popup dialog that allows you to select the libraries to be searched, and then performs the search, displaying a progress bar.

The option writes results to the Find and Replace log, and then opens the Find and Replace window when it has completed the search. Note that the option does not search calculations stored with objects, it only searches method lines.

Calls located may not be actual calls to the method, since for example calls like item.$method cannot be resolved, so if the call occurs in the correct class (or in the inheritance hierarchy), the call will be treated as found.

Debugger Debug Panel

The debugger Debug Panel has been moved to the top of the code editing part of the Method editor; in previous versions it was located in the window title bar. The new debugger command panel contains the Go, Step in, Step Over, etc commands, but the Remote debugger command panel does not include the "to line" command. The last command on the panel is the "clear stack" option.



For the local debugger, there is a new menu item on the method editor View menu, **Show debug command panel** to show the panel (which is enabled by default). The old debug toolbar options are now initially hidden, but you can show them using the **Toolbar…** menu item on the method editor **View** menu.

For the Remote debugger, the debugger command panel replaces the toolbar buttons that previously provided commands.

Where the toolbar configuration has been saved in omnis.cfg, you need to use the restore defaults button on the Toolbar… configuration window, to set the toolbar buttons to their new default state.

Overriding or Inheriting multiple methods

The Method editor can now override or inherit multiple methods in a single operation. You can multi-select the methods from the class Methods list or a single object and then override or inherit the methods, as appropriate, from the context menu or the **Modify** menu. (ST/CE/220)

If all selected methods are built-in or inherited, they can be overridden with a single override method command on the context menu, or the Modify menu.

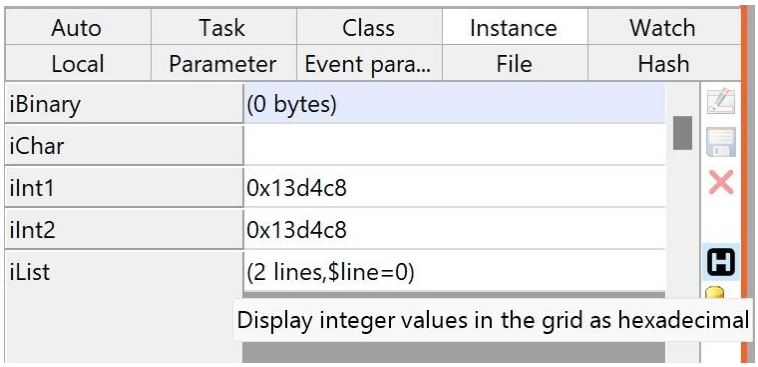
If all selected methods override inherited methods, they can be deleted and re-inherited using a single inherit method command on the context menu, or the Modify menu.

If all selected methods override built-in methods, they can be deleted and set back to built-in using a single built-in method command on the context menu, or the Modify menu.

In addition, the options **Inherit variables...** and **Override variables...** have been added to the Variable panel context menu to allow multiple variables to be inherited or overridden in a single operation. These commands are only present for subclasses, for task, instance and class variables. (ST/CE/221)

Display Integers as Hex

There is a new option in the debugger Variable panel (in both the Method Editor and Remote debugger), to display integer variables as hex rather than decimal – the new option is available by clicking the H icon in the panel sidebar.



In addition, in either display mode (decimal or hex), when modifying an integer, you can either enter a decimal value or a hex value.

Code Assistant

Parameter list order

The ordering of variable and parameter names in the Code Assistant help list has been improved. (ST/DB/1349)

The sorting is now in the following order: variables or names, functions, notation, constants, events (sorted alphabetically within each set).

There is a new config item ‘oldSortOrder’ in the ‘codeAssistant’ section of config.json, which is false by default; set this to true if you want to use the old sorting.

Do command

Code assistance for the Do command and its variants has been improved. (ST/DB/1392)

The Code assistant now adds matching commands when entering what could be command parameters, e.g. when entering probable parameters for Do, the Code Assistant will also add Do method, etc, commands to the list.

Note that when typing command names, you can omit spaces and the command will be found more easily, for example, to find *Do inherited,* you can type doi.

$obj and $field

Code assistance for $cinst.$obj and $cinst.$field has been improved. (ST/DB/1391 and ST/JS/2193)

Code assistance has been added for $cinst.$field to allow code assistance to be provided without making $cinst.$objs harder to enter; $cinst.$field is equivalent to $cinst.$obj, but $field only works for subform and subwindow instances.

The notation $field behaves the same as $obj when used with a subform or subwindow instance, for remote forms and windows only. This allows code assistance to be better targeted, and also prevents $obj from taking over as the first property in the code assistant list after typing “$cinst.$o”.

Code Assistant Width

There have been a few enhancements in the size and format of the Code Assistant help popup window. There is a new entry **width** in the ‘codeAssistant’ section of the config.json file to allow you to set the width of the code assistant window (the default is 768 pixels); the value must be between 512 and 1536 inclusive.

As a consequence of the enhancements, the items drawTurnedOverPanelTextAtBottom, helpPanelHeight, and minWidth have been removed from the ‘codeAssistant’ section of config.json.

Item Reference Classes

You can now use an existing library class as an item reference class, and the Code Assistant will show the instance notation for the selected class. (ST/BE/1419)

When creating an **Item Reference** variable in the Method Editor, the variable subtype dialog (that allows you to enter the class for a variable) now has two tabs at the bottom: Generic and Instance:

- **Generic** allows you to enter a class as previously supported.
- **Instance** allows you to enter a class of the form @classname or @library.classname (these new classes are identified by a leading @ character).

An item reference that uses a class in this way provides code assistance for both the built-in methods and properties of an instance of the class, as well as user methods for instances of the class.

In addition, there is an entry field that allows additional notation to be appended, e.g. $objs.objname (this field has code assistance; the additional notation must start with a $). For example, if you have a field named *test* in window class *myWindow*, you can enter $objs.test in the entry field, and select the window class from the tree. This results in an item reference class @myWindow.$objs.test. When using the variable, code assistance is for this field, so you get both the built-in methods and properties and the user methods for the field.

Jump to Variable Definition

You can now jump to a variable definition in the Variables pane in the method editor using the **Variable** context menu; this is useful if a method contains many variables and saves you visually scanning the variable list to find the variable. (ST/DB/1335)

To show the definition for a variable, Right-click on the variable name in your code and select the **Variable Definition** option. The focus will jump to the appropriate tab in the Variables panel, highlighting the variable. Alternatively, you can select the **<Variable-Type> variables** option to pop up a separate Variables panel highlighting the variable.

The highlighting of the current variable grid line when the variable grid has the focus has been enhanced, so that the current line is highlighted for both inherited variables, and all variables when the class is read-only.

Jump to Search or Error Item

When you search for an item in the Code Editor, instances of the found item are shown on the window scroll as a green marker. Similarly, errors in the code are marked in the scrollbar as a red marker.

On Windows, you can Ctrl-click in the scrollbar to jump to that position in the code text, i.e. Ctrl-clicking on a find or error marker goes to the search item or error in the code text. (ST/DB/1289)

On macOS, the general system preference for scrolling can be set to Jump to the position that has been clicked, or you can Alt+click to achieve the same thing (this is the case for existing versions of Omnis).

Variable Names

Renaming Inherited Variables

Renaming an inherited Class or Instance variable in the superclass now shows an optional warning dialog (from which Find and Replace can be opened) if there are subclasses that may be using the variable. (ST/VR/332)

Naming Variables

The description for reporting errors when naming a variable has been improved, to ensure the correct naming of variables (ST/DB/1321). The description is now:

Each character in the name must either be a Unicode alpha character, a decimal digit, or a character in the range U+80 to U+ff inclusive. The first character cannot be a decimal digit.

Inherited Descriptions

Method description and notes are now inherited from the nearest superclass of the current class, as long as the description or notes of the current class is empty and there is a description in the superclass. (ST/DB/1411)

The Method Editor now shows the inherited or built-in method description/notes for a method that is either inherited, built-in, or overridden, when the description/notes in the current class is empty. To indicate that these are inherited, they are drawn in the inherited color or built-in color, as appropriate. If you want to change the value for an overridden method, you can right-click on description/notes, and select "Make Editable", and edit the value. You can revert to the inherited value by editing and deleting the description/notes; when the focus leaves the edit field, it shows the inherited value.

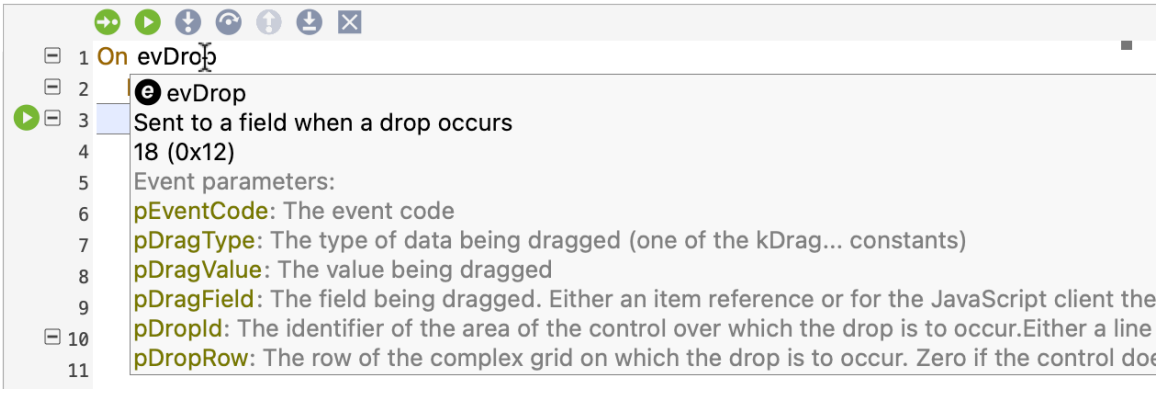
Object Search

When you use the Search box in the Method editor (and remote debugger) the search will now include *Object names* as well as class method names. (ST/WC/576)

The new search behavior allows you to locate controls and other objects in the method tree list, such as containers or text labels, in order to display and edit their methods. The setting for this search behavior is stored in a new item ‘includeObjectNodesInTreeSearch’ in the ‘methodEditorAndRemoteDebugger’ group in the config.json file (true by default).

Event Parameters

Event parameter names and descriptions have been added to the tooltip for an event, displayed when you hover the pointer over the event name. (ST/EM/230)



Break On Event Option

A new **Break On** option has been added to the Debug menu in the Method Editor to allow you to select which events will stop the debugger while debugging remote form and window instances. (ST/EM/228)

There is a new hierarchical menu item Break On in the Debug menu that opens a window that allows you to select the events that will cause a debugger break when they are generated in the relevant instance type. IDE windows do not cause a break.

The new option replaces the ‘Debug Next Event with Break On Selected Window Instance Events’ and ‘Break On Selected Remote Form Instance Events’ options.

Copy Method Name

The **Copy Name** command has been added to Method Editor tree context menu to enable you to copy the name of a method. (ST/CE/165)

Edit List Line

The **Edit Line In New Window** option has been added to the context menu for the List variable window (right-click in the left-most column) to allow you to open a separate editor grid to view or edit the data for a list line. (ST/DB/1374)

JavaScript Error Messages

JavaScript code generation and JSON import-export error messages now go to the Trace log rather than the Find and Replace log in a Runtime/Server version of Omnis. (ST/JS/2821)

Method Editor Focus

A new option has been added to make sure the focus is on Method name list when the Method Editor is opened via a modify command, for example, by opening the Method Editor from the Studio Browser. (ST/DB/1419)

The option is **modifyMethodsCommandSetsFocusToTree** in the 'methodEditor' section of the config.json file. The option defaults to false, meaning the behavior of previous versions is maintained.

System Notifications

Omnis can now send notifications to the operating system on the end user's computer, on both Windows 10/11 and macOS. You have control over the content of notifications and when they are sent via your Omnis code using a new **ONOTIFY** object. When sent, a notification will pop up on the end user's screen and will be added to the **Notification Center** for the current operating system. (ST/HE/1686)

The end user can click on a notification and either start Omnis, or if Omnis is already running, bring Omnis to the front. In both of these cases, the method **$localnotify()** in the **Startup_Task** (in the library that sent the notification) receives parameters specific to the notification and the method can then process the click, or call another method, for example.

As well as sending notifications, there are additional functions that allow you to add a badge to the application icon to alert the end user about the notifications.

There are two interfaces provided to send a system notification:

- **An object,** providing a way to encapsulate notification parameters.
- **A function,** providing a simple interface to send a notification with a single line of Omnis code.

As with many features in Omnis, these interfaces provide a single, cross-platform method to interact with system notifications on both Windows and macOS.

Notification Object

The **Notification Object** provides a way to encapsulate notification parameters. To use the object, you need to set the *Subtype* of an *Object variable* to the **LocalNotify** external object. The object has the following properties:

Property	Description
$action	A value that specifies up to 2 optional actions that are to be included in the notification; on Windows, this is via one or two buttons; on macOS, this is either via a button for a single action, or via an

Property	Description
	options popup for two actions. A ‘Specifying Actions’ section
$delay	The delay in seconds between the call to $sendlocal() and the notification being delivered. Omnis can quit before the notification is delivered, as the operating system takes care of deferred delivery
$messageimage	Image(s) to be displayed with the notification. See the ‘Specifying Images’ section
$messagetext	The text of the notification. This is the main notification message, displayed in a plain font. The operating system will truncate this if it occupies more than 4 lines, either due to word wrapping, or the presence of newline characters (kCr, lLf or kCr kLf)
$notifylib	See section ‘Handling Notification Clicks’ for details about this property
$title	The title of the notification. Some text, displayed in bold font above the main notification text. The operating system will truncate this if it is too long. Windows allows this to occupy two lines, if you separate the lines using either kCr, lLf or kCr kLf. macOS only allows a single line
$userinfo	A row containing user information that is passed to the $localnotify() method when the user clicks on the notification or a notification action. It must be possible to convert $userinfo to JSON. See section ‘Handling Notification Clicks’

To send a notification, created using the current property values, use the $send() method of the object.

```
Do Object.$send([&cErrorText])
```

Sends a local operating system notification using the current property values. The parameters are as follows:

Parameter	Description
cErrorText	A character variable that receives text describing an error if $send() fails

If the call to $send() fails, it returns the value #NULL, and sets the cErrorText parameter if it is provided.

If the call to $send() succeeds, it returns a character string. This is a string that uniquely identifies the notification. You can use this string to remove the notification from the system Notification Center, if for example the notification is no longer relevant.

Notification Functions

The **ONOTIFY.$sendlocal()** function sends a system notification.

```
ONOTIFY.$sendlocal(cTitle,cMessage,vImage,iAction,wUserInfo,[iDelay=0,&cErrorT
   ext])
```

The parameters are as follows:

Parameter	Description
cTitle	The title of the notification. Some text, displayed in bold font above the main notification text. The operating system will truncate this if it is too long. Windows allows this to occupy two lines, if you separate the lines using either kCr, lLf or kCr kLf. macOS only allows a single line
cMessage	The text of the notification. This is the main notification message, displayed in a plain font. The operating system will truncate this if it

Parameter	Description
	occupies more than 4 lines, either due to word wrapping, or the presence of newline characters (kCr, lLf or kCr kLf)
vImage	Image(s) to be displayed with the notification. See the 'Specifying Images' section
iAction	A value that specifies up to 2 optional actions that are to be included in the notification; on Windows, this is via one or two buttons; on macOS, this is either via a button for a single action, or via an options popup for two actions. A 'Specifying Actions' section
wUserInfo	A row containing user information that is passed to the $localnotify() method when the user clicks on the notification or a notification action. It must be possible to convert $userinfo to JSON. See section 'Handling Notification Clicks'
iDelay	The delay in seconds between the call to $sendlocal() and the notification being delivered (optional). Omnis can quit before the notification is delivered, as the operating system takes care of deferred delivery
cErrorText	A character variable that receives text describing an error if $sendlocal() fails

If the call to $sendlocal() fails, it returns #NULL, and sets the cErrorText parameter if it is provided.

If the call to $sendlocal() succeeds, it returns a character string. This is a string that uniquely identifies the notification. You can use this string to remove the notification from the system Notification Center, if for example the notification is no longer relevant.

Specifying Images

You can specify an image for the notification via the $messageimage property of the object, or vImage parameter of the function. macOS only allows a single image, whereas Windows allows up to three. The Windows images must each have an associated type, and there can only be one image of each type. The image types are identified by constants:

Constant	Description
kONOTIFYimageTypeNormal	The image is to be displayed below the notification.
kONOTIFYimageTypeLogo	The image is to be used as the application logo.
kONOTIFYimageTypeHero	The image is to be used as the hero image (this is Windows terminology). This is an image displayed across the top of the notification, and it must have the size 364x180 (728x360 retina) to look good, otherwise the system resizes it and crops it.

Images can be specified either using a character variable, or by using a list. To include no image in the notification, either use an empty character variable or value, or use a list with no lines and the correct number of columns (see below).

If you use a character variable, with a non-empty value, the notification has a single image; the character variable must contain the full pathname of an image file (typically PNG or JPEG), and on Windows it will have the type kONOTIFYimageTypeLogo.

If you use a list variable, then the list must have at least one column on macOS, and at least 2 columns on Windows. The number of rows is limited to 1 on macOS, and 3 on Windows (one for each type). Column 1 of the list contains the full pathname of an

image file (typically PNG or JPEG), and column 2 contains a kONOTIFYimageType… constant.

The system is responsible for laying out the notification content (i.e. you have no control over layout), and you should avoid using very large images in a notification.

Specifying Actions

You can specify up to 2 actions to be included with the notification. To specify no actions, the action value can be either zero or kONOTIFYactionNone.

The actions are pre-defined, as macOS requires actions to be pre-defined. To specify one or more actions, use the following constants, which can be added together when specifying 2 actions:

Constant	Description
kONOTIFYactionAccept	The notification displays the Accept action.
kONOTIFYactionClose	The notification displays the Close action.
kONOTIFYactionDecline	The notification displays the Decline action.
kONOTIFYactionDelete	The notification displays the Delete action.
kONOTIFYactionNo	The notification displays the No action.
kONOTIFYactionOpen	The notification displays the Open action.
kONOTIFYactionPrint	The notification displays the Print action.
kONOTIFYactionYes	The notification displays the Yes action.

Handling Notification Clicks

By default, when the user clicks on either a notification, or a notification action, Omnis executes the method **$localnotify()** in the **Startup_Task** of the library containing the code calling ONOTIFY.$sendlocal() or object.$send().

When using a LocalNotify object to send the notification, you can override the library using the **$notifylib** property; this property is the internal name of the library whose startup task is to receive the $localnotify() call. If you do not assign $notifylib, or set it to empty, the default behavior applies.

If Omnis is not running when the user clicks on either a notification or a notification action, the system starts Omnis. Omnis defers calling $localnotify() until the startup task has completed, to allow startup libraries to be opened and their initialization to complete.

If Omnis is running when the user clicks on either a notification or a notification action, the system brings Omnis to the front.

When the system calls Omnis to tell it about a notification, and the library in which $localnotify() is to be called is not open (after waiting for startup to complete if necessary), Omnis ignores the call.

$localnotify appears in the built-in methods of a task class, so you can override it. It has 2 parameters:

Parameter	Description
pAction	A kONOTIFYaction… constant that identifies the action pressed by the user. kONOTIFYactionNone (zero) if the user clicks directly on the notification, rather than a button or popup.
pUserInfo	A row. The user info value that was supplied when sending the notification.

$localnotify() is not required to return a value.

Removing Notifications

The notification object and function send methods (Object.$send() and ONOTIFY.$sendlocal()) both return a **unique id** to identify the notification that was sent. If you want to remove the notification from the Notification Center at some point later (possibly after restarting Omnis), you need to save the id somewhere, e.g. in a local SQLite database.

To remove one or more (or even all) notifications sent by Omnis, use the method:

```
ONOTIFY.$removelocal([vIDs,&cErrorText])
```

The parameters are as follows:

Parameter	Description
vIDs	Either a single character id, or a single column list of ids, to remove. To remove all local notifications sent by Omnis, pass an empty character string, a list with no lines, or omit the vIDs parameter.
cErrorText	A character variable that receives text describing an error, if $removelocal() fails.

If the call to $removelocal() fails, it returns the Boolean value false, and sets the cErrorText parameter if it is provided. If the call to $removelocal() succeeds, it returns the Boolean value true.

Badges

ONOTIFY provides functions that allow a badge to be added to, or removed from, the application icon.

On Windows, this applies to the application icon in the taskbar. On macOS, this applies to the application icon in both the dock, and the task switcher. The two operating systems behave differently, because of the way their APIs work.

The badge is only displayed while Omnis is running.

$setbadgecount

ONOTIFY.$setbadgecount(iCount[,&cErrorText,iBadgeColor,iBadgeTextColor])

Sets the application icon badge to the specified count. The parameters are as follows:

Parameter	Description
iCount	The count to display as the badge. Must be greater than zero. When running on Windows, a value greater than 99 is displayed as 99+.
cErrorText	A character variable that receives text describing an error, if $setbadgecount() fails
iBadgeColor	Windows only. The background color of the count badge. Defaults to styledbadgebackgroundcolor in the system section of appearance.json.
iBadgeTextColor	Windows only. The text color of the count badge. Defaults to styledbadgetextcolor in the system section of appearance.json.

Note that the appearance.json items styledbadgebackgroundcolor and styledbadgetextcolor have been moved to the 'system' section of appearance.json.

If the call to $setbadgecount() fails, it returns the Boolean value false, and sets the cErrorText parameter if it is provided. If the call to $setbadgecount() succeeds, it returns the Boolean value true.

$setbadgeicon

ONOTIFY.$setbadgeicon(vIconId[,&cErrorText,iBadgeColor=kColorHilight])

Note this is available on Windows only. Sets the badge on the application icon to be the specified icon. The parameters are as follows:

Parameter	Description
vIconId	The icon id of the icon to display as the badge. The size component is ignored, as badges are always 16x16.
cErrorText	A character variable that receives text describing an error, if $setbadgeicon() fails
iBadgeColor	The color to be applied to the themed SVG; only applies if the icon is a themed SVG. Default is kColorHilight.

If the call to $setbadgeicon() fails, it returns the Boolean value false, and sets the cErrorText parameter if it is provided. If the call to $setbadgeicon() succeeds, it returns the Boolean value true.

$removebadge

ONOTIFY.$removebadge([&cErrorText])

Removes the badge from the application icon. The parameters are as follows:

Parameter	Description
cErrorText	A character variable that receives text describing an error, if $removebadge() fails.

If the call to $removebadge() fails, it returns the Boolean value false, and sets the cErrorText parameter if it is provided. If the call to $removebadge() succeeds, it returns the Boolean value true.

Enabling Notifications

To receive notifications from Omnis, notifications have to be enabled for Omnis in the respective system settings. On **Windows,** you can enable System Notifications via the Settings >> System dialog, then the **Notifications & Actions** option. On **macOS,** you can use the System Preferences >> **Notifications & Focus** option.

The following describes how Omnis is identified by each operating system in order to initialize system notifications.

macOS

The macOS operating system identifies applications using their application bundle identifier, so if you install multiple versions of Omnis on the same macOS system, notification settings, such as those in the **Notifications & Focus** section of System Preferences, apply to all applications with that bundle identifier.

For Studio 11.0, the application bundle identifier now includes the version, that is, net.omnis.omnisStudio.11.0. In addition, the Development, Server, or Runtime versions of Omnis are identified by type. Therefore, the application bundle identifier is net.omnis.omnisStudio.<type>.11.0 where <type> can be Dev, Server, or Run, so these three executables can co-exist on the same macOS system. In addition, the deployment tool caters to the different types.

Windows

For notifications to work on Windows, and in particular to allow clicks on notifications to be passed to Omnis, Omnis needs to register an AppUserModelID and store the AppUserModelID in a shortcut to Omnis in the system Start menu.

There are two configuration entries in the ‘windows’ section of config.json:

Entry	Description
initLocalNotifications	Boolean. Default true. If true, Omnis initialises the interfaces required to send notifications to the local Notification Center.
createShortcut	Boolean. Default true. If true, and there is no shortcut to itself in the Start menu, Omnis creates a shortcut to itself in the Start menu. It then modifies the shortcut to contain the AppUserModelID required for local notifications to work.

Omnis uses core resource string 9 as the template for its AppUserModelID. This defaults to "OmnisSoftware.OmnisStudio.$.11". To create the AppUserModelID, Omnis replaces $ with Dev, Server or Run to identify the Development, Server or Runtime version of Omnis.

The deployment tool (Windows only) allows you to customize resource 9. Note that if there is no $ placeholder in the resource, the resource value is not changed by the attempt to insert Dev, Server or Run.

Power Management Notifications

Omnis Studio can receive sleep and wake notifications from the operating system to indicate power management changes: the following enhancements apply to macOS and Windows.

Requests from the system to go into idle sleep, when there is no user activity, can be denied on macOS or disabled on both macOS and Windows.

This allows the system to remain awake if Omnis Studio is busy.

Power Management Methods

Each task has a set of power management methods which can be overridden.

$systemcansleep (only sent on macOS)

All library task instances receive a call to the **$systemcansleep** method when the system is requesting permission to go into idle sleep.

If all instances of this method return kTrue then sleep will be allowed to continue and there will be a subsequent call to **$systemwillsleep**.

If any instance returns kFalse from this method then sleep will be aborted.

The total time taken to return from all calls to this method must not exceed 30 seconds or the sleep will continue.

$systemwillsleep

All library task instances receive a call to the **$systemwillsleep** method when the system is starting a sleep which cannot be cancelled, e.g. low battery or laptop lid is closed. This is delivered before any hardware is powered off.

The total time taken to return from all calls to this method must not exceed 30 seconds on macOS or 2 seconds on Windows otherwise the sleep will continue.

This call can be used by an application to save the state before the system sleeps.

Operations can be performed such as saving data to disk or disconnecting from databases.

$systemwillwake

All library task instances receive a call to the **$systemwillwake** method when the system is beginning to power on, i.e. most hardware has not been powered on. Attempts to access disk, network, the display, etc. may result in errors or blocking the process until those resources become available.

On Windows once user interaction is detected, e.g. mouse or keyboard input, then the system will send **$systemdidwake**.

$systemdidwake

All library task instances receive a call to the **$systemdidwake** method when wakeup has completed and the system is powered on. This call can be used by an application to resume the state which was saved when the system went to sleep. Operations can be performed such as loading data from disk or reconnecting to databases.

Disabling idle sleep

Typically the system will be setup to sleep after a set period of inactivity. An Omnis application can disable this by using the **$disablesystemidlesleep** root preference. If set to kTrue the system will be prevented from going into idle sleep.

An application will still receive a call to the method **$systemwillsleep** if the system is starting a sleep which cannot be cancelled.

On macOS the system will log a message to the system log to indicate the reason why the system is blocked from going into idle sleep.

A Studio application can set this log message by using the $disablesystemidlesleepreason root preference.

The default for this message is set to 'Omnis Studio is busy' but can be altered by editing the string for resource number 1835.

The message should describe the name of the application and the activity blocking the sleep, e.g. "MyApp is searching appointments".

Window Components

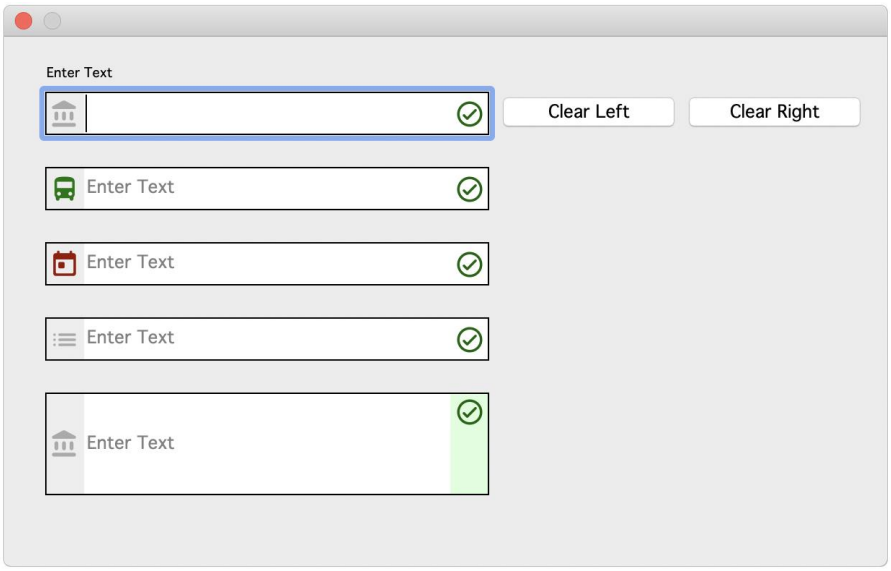
Entry Fields

Field Border Icons

You can now add icons to the left and right border of Entry fields to provide a visual hint or feedback, adding to the ease of use for the end user. For example, you could show a check mark icon to indicate when a field has been correctly filled out. (Note that field border icons are not available for JS Edit controls.)

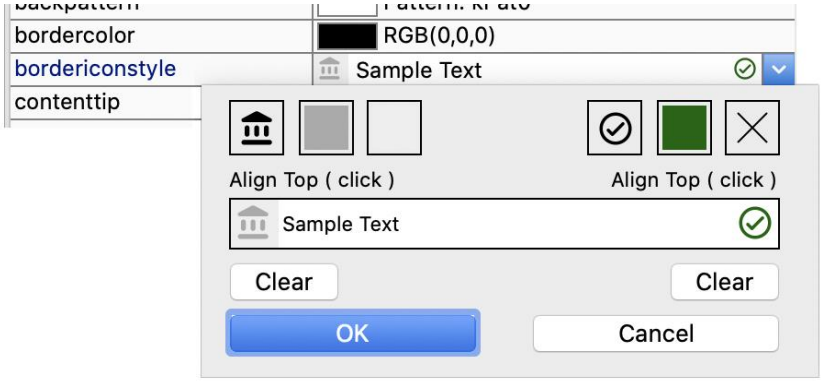
There is a new example app called **Field Border Icons and Content Tips** in the **Samples** section of the **Hub** in the Studio Browser (note there is a New option to display the new examples only).

Field border icons can be added to all window class entry field types, including *Single Line Entry fields, Multi Line Entry fields, Masked Entry fields,* and *Token Entry fields*; the new property only applies when the field border style in $effect is set to kBorderCtrlEdit (that is, when $fieldstyle is the default CtrlEditText), kBorderCtrlList, or kBorderPlain. Note that the icons are for display only, and do not report any events, so for example, you cannot add code to them to react to clicks.



You can set the color of the icon (if using an SVG), the background color of the icon area, and the vertical alignment. By default the background color is the same as the control. The content area of the edit field is adjusted inside the frame to accommodate icons if set. If the control is below 20 pixels in height the icon will scale down.

The new property **$bordericonstyle** stores the left and right icon configurations for an Entry field, including the iconid of the left and right icon (which can be a SVG or PNG specified by character or integer iconid), plus the icon color and background color (e.g. kColorDefault or a RGB value). The single property stores the settings for the left and right icons, which you can specify in the Property Manager:



There is a new method **$setbordericonstyle** that allows you to set the left or right icon for an entry field, or clear the icon(s); the method has the following syntax:

- **$setbordericonstyle**(bLeftIcon[,cIcnIDName,iIcnTintColor,iBackTintColor])
  bLeftIcon should be kTrue to enable a Left icon, or kFalse for a Right icon
  cIcnIDName is the name ID of the icon (can be a string for a SVG icon)
  iIcnTintColor, iBackTintColor are the colors for the icon and background

The following code for a field event method shows a warning icon on the right if no content is added, otherwise if content is added a check mark icon is shown:

```
On evAfter
   If len($cobj.$contents)<=0
      Do $cobj.$setbordericonstyle(kFalse,"cancel",kDarkRed)
      Sound bell
      Quit event handler (Discard event)
   Else
      Do $cobj.$setbordericonstyle(kFalse,"check_circle",kDarkGreen)
   End If
```

To clear an icon, you pass bLeftIcon as either kTrue (left icon) or kFalse (right icon) with no value for cIcnIDName, as follows:

```
Do $cinst.$objs.FIELD.$setbordericonstyle(kTrue) ## clears the left icon
```

Content Padding

The $contentpadding property has been added to Entry Fields to allow you to add padding around content inside a window class Entry Field. (ST/HI/2017)

The $contentpadding property is specified in pixels, with 1 to 4 pixel values separated by -, in the order left, top, right, bottom; if a single value is specified it is applied to all four sides. If the bottom value is omitted, the top value is used. If the right value is omitted, the left value is used. If the top value is omitted, the left value is used. For example, a value of '3-2-1' gives a 3 pixel gap on the left, a 2 pixel gap on the top and bottom, and a 1 pixel gap on the right.

Animated Content Tips

Content tips for entry fields can now be animated, meaning that when enabled and the focus enters the field, the content tip inside the field will 'float' or move up above the field, which can be very useful for providing help to end users and creates a more interactive UI. There is a new example app called **Field Border Icons and Content Tips** in the **Samples** section of the **Hub** in the Studio Browser.

To allow animated content tips, the **$animateui** property has been added to the window class Entry field, Multi-line Edit field, Token Entry field, and Combo box (the entry field part). When set to kTrue for these field types, and when the focus enters the field, the content tip will float above the field. When animated, the content tip will shrink to 80% of the edit field font size, and use the same font colors as the edit field.

This feature is not supported for Entry fields or Combo boxes that are inside a Complex grid.

Strip Control Characters from Edit Fields

The $pastestripscontrolcharacters property has been added to window class Edit Fields (Single-line entry), Multi-line Edit fields, Masked Entry fields, and Token Edit fields (as well as Combo boxes, Data grids, and String grids). The new property is also a library preference. (ST/HI/1983)

If $pastestripscontrolcharacters is true (for a control or all controls via the library preference) then all unused control characters are removed when pasting character data into the edit field, or the editable part of combo boxes, data grids, or string grids.

Control characters are 0-0x1f and 0x7f. All control characters are "unused" except for the carriage return line delimiters used by certain fields.

The library preference $clib.$prefs.$pastestripscontrolcharacters defaults to kTrue in a new library, and kFalse in existing (converted) libraries.

Emoji and Symbols in Edit Fields

On macOS, support for the standard Emoji and Symbols menu item from the Edit menu has been added to the window class Entry field. This will display the Character Viewer to allow entry of emoji, symbols, accented letters and characters from other languages. (ST/WO/2603)

The Emoji and Symbols menu item is available by default. To remove it, set the new item **useCharacterPalette** in the 'macOS' section of the config.json file to false. Note this does not support drag and drop from the character viewer into Studio.

Token Entry Field

The events **evTokensAdded** and **evTokensDeleted** have been added to the Token Entry Field. In addition, tokens can now have a tag, and the method $gettokens has a new optional parameter. (ST/EM/231 & ST/NT/793)

Token Tags

Tokens in the Token Entry Field can now have a **tag.** A tag is a character string that the application can use to identify the source of the token, or some other information about the token. Although the tag is part of the data, it is not visible to the user.

To use tags, set the new property $tokentagseparator, which is a single character that separates the token tag from the rest of the data for a token. The property defaults to empty, meaning tags will not be used. You can enter \t and \f to use chr(9) and chr(12) respectively. Using ~ as the tag separator, each token can now be one of the following, depending on whether tags are being used:

- tokenValue
- displayText<tokenValue>
- tokenValue~tag
- displayText<tokenValue>~tag

You should ensure that the tag separator and < > characters used when including display text are not part of the displayText or tokenValue.

The characters \t and \f can be entered in the $tokendelimiters to use chr(9) and chr(12) respectively. Using \t or \f allows the unambiguous use of JSON syntax characters in tags.

There is a new property $canedittokens which defaults to kTrue for compatibility. When kFalse, the end user cannot edit a token as text (by double clicking on it, or pressing return when it is selected); however, you can type some text to cause the token popup menu to display. When using a tag in the token, data you would typically set $canedittokens to kFalse, because otherwise the tag could become meaningless if the user edits the token data.

Token Events

There is new property $sendtokenevents which defaults to kFalse. When kTrue, the control sends the events evTokensAdded and evTokensDeleted.

- **evTokensAdded**
  sent to the token entry field when one or more tokens have been added (if $sendtokenevents is true)
- **evTokensDeleted**
  sent to the token entry field when one or more tokens have been deleted (if $sendtokenevents is true)

Both of these new events have an event parameter, pTokenChanges which is a list of tokens that have been added or deleted, comprising two or three columns: name (token), display (display text) and optionally the tag.

Get Tokens

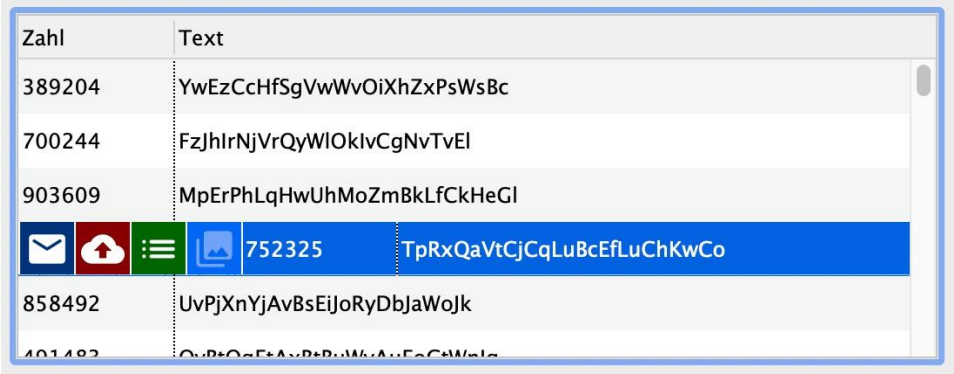
The $gettokens method has a new optional parameter, bSplit (defaults to kFalse for compatibility). When passed as kTrue, the returned list has two or three columns, name, display and optionally the tag. The bIncludeDisplayString parameter is ignored if bSplit is true.

List Row Buttons

You can now add a set of buttons to the **left** and/or **right** side of a row in a standard List or Headed List control. The buttons would typically act on the data in the row, such as opening another window to edit the row data, or deleting the row.

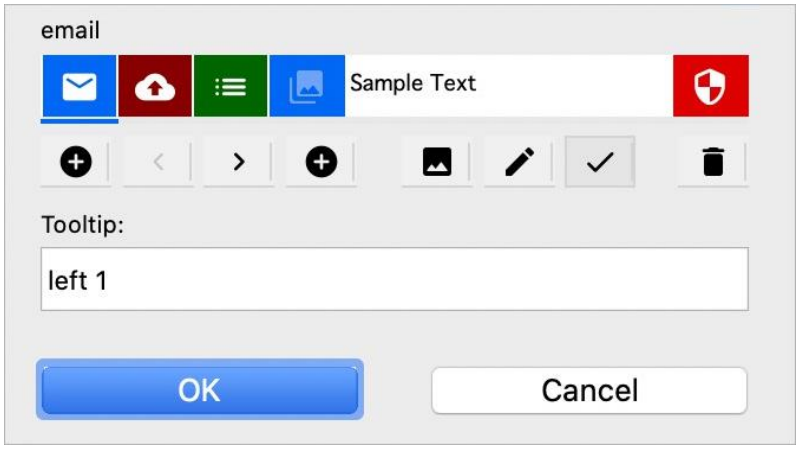
There is a new example app called **Row Buttons** in the **Samples** section of the **Hub** in the Studio Browser (note there is a New option to display the new examples only).

The row buttons slide out as the mouse enters the left or right side of the current row in the list, or as the Shift+Control+Left or Right arrow keys are pressed when a line in the list is selected. Selecting a button closes the whole row of buttons, and the name of the selected button is sent to the event method for the list. Row buttons work best with a larger row height (font size for the list), or when $linehtextra is set in a List or Headed list control.



Adding Row Buttons

List controls and Headed list controls have a new property **$rowbuttons** which stores the definition for the left and right buttons in the list row. When you select the property in the Property Manager in design mode a dialog opens allowing you to specify the buttons for the row. The button definition stored in $rowbuttons will be applied to every row in the list.



In the edit dialog, you can add a left or right button using the left or right + icons. Clicking on a button makes it current (shown underlined) allowing you to change its attributes. You can move a button in the row order using the left and right arrow buttons.

The image icon allows you to add an icon, which is an SVG image from an icon set (PNG is not supported); the name of the icon becomes the name of the button, which is shown in the top-left of the dialog, and used in the notation to reference the button. The pencil icon allows you to set the background color of the button. The check icon allows you to disable the button. You can also add a tooltip for a button.

You can delete a button by selecting it and clicking the trash icon.

Setting Row Buttons

There is a new method **$setrowbuttons** which allows you to specify the left or right buttons for a list row at runtime, giving you more control over the buttons for individual list rows; for example, you can call this on the evClick event for the list.

The definition for $setrowbutton is as follows:

- $setrowbutton(*bLeftIcon*,[*cIcnIDName, iTintColor, bDisabled, cTooltip*]) adds a row button.
  bLeftIcon: kTrue for a left button, kFalse for a right button
  cIcnIDName: the name of an SVG icon for the button
  iTintColor: color of the button background
  bDisabled: kTrue if the button is disabled
  cTooltip: the tooltip for the button

You can clear the whole row of buttons by sending *bLeftIcon* as either kTrue or kFalse without any further parameters.

```
Do $cobj.$setrowbuttons(kTrue)    ## clears the left buttons
Do $cobj.$setrowbuttons(kFalse)   ## clears the right buttons
```

The following will add a button:

```
# adds a button only for row 2
If $cobj.$line=2
    Do $cobj.$setrowbuttons(kTrue,"language",kDarkRed)
End If
```

Events

The evRowButtonClicked event is sent to the list control when a row button is selected, and pRowButton will contain the icon name of the button clicked.

```
On evRowButtonClicked
    If pRowButton="language"
        # discards the event for a button with the icon 'language'
        Quit event handler (Discard event)
    End If
    # process the button click
```

List Box

Selected List Line Colors

The $selectiontextcolor and $selectionbackcolor properties have been added to all window class List components, including standard List boxes, Icon arrays, Headed lists, Checkbox lists, Complex grids, and Tree lists. (ST/GR/409)

The $selectionbackcolor property is the back color of selected lines. kColorDefault means use the default color. When not kColorDefault the specified color only applies when the control has the focus.

The $selectiontextcolor property is the text color of selected lines. kColorDefault means use the default color. When not kColorDefault applies irrespective of whether the control has the focus.

Extra Line Height

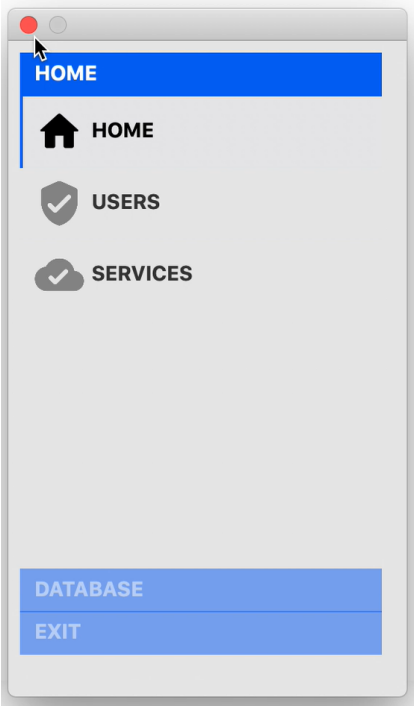
The $linehtextra property has been added to the List box control, which is the number of extra pixels added to the height of each line (in a list box or headed list). (ST/WO/2739)

Tab Strip

Tab Strip Groups

The **Tab Strip** control now supports groups, which means you can arrange the tabs vertically into separate groups. The new group support only applies when the Tab Strip is in ***vertical mode*** with one of the animated styles: kTabStripAnimSquare, kTabStripAnimLine, kTabStripAnimDot, or kTabStripAnimRndSquare.

There is a new example app called **Tab Groups** in the **Samples** section of the **Hub** in the Studio Browser (note there is a New option to display the new examples only).



The individual tabs are specified as a comma-delimited list in the $tabs property, as in previous versions. Each tab has a new property, $tabgroupname (under the Pane tab in the Property Manager), which specifies the group the tab belongs to, and $tabgroupcolor specifies the color used for the background of the tab group header. When you specify a group name in $tabgroupname, the group is added to the Tab strip and the tab is added to the group automatically.

The evTabGroupChanged event is sent with the pTabGroup parameter when the control is about to change groups, which can be discarded if required.

Keyboard Navigation

When grouped mode is enabled and the Tab Strip has the focus, you can use the keyboard to navigate the tabs or groups: the Up and Down keys navigate between the *tabs* within the group order (the groups will open automatically as you navigate to a tab), while the Shift Up and Down keys will jump up and down between groups, opening the group as it gets the focus.

Animated Line and Dot Modes

The color specified in $tabcolor is now used to show the non-selected lines and dots for Animated Line and Animated Dot modes for Tab Strips. (ST/HE/1860)

The $tabcolor property specifies the color to be used for the line path or placeholder dots for non-selected tabs when $squaremode in a Tab Strip is set to kTabStripAnimLine or kTabStripAnimDot. In the following example, tab 4 is selected and the non-selected lines and dots are shown under tabs 1 to 3 since $tabcolor is set to kColor3DShadow.

kTabStripAnimLine

kTabStripAnimDot

By default the color in $tabcolor is kColorDefault which means the line path or placeholder dots are not shown (as in previous versions), so you need to select a color in $tabcolor to show the non-selected lines and dots.

Selected Tab Text Style

You can now set the font style of the selected tab in a **Tab Strip** window control. (ST/HE/1861)

A new property **$selectedtabtextstyle** has been added to the Tab Strip window class control that allows you to assign the font style of the currently selected tab, overriding the font style for all tabs set in $fontstyle (this new property applies to all modes except kTabStripOriginal).

Round Check Boxes

Check boxes can now be displayed using a *Round style,* rather than the standard square box image. (ST/WO/2632)

The $buttonstyle property for Check boxes can now be set to **kCheckBoxRound.** The style will create the maximum possible circle within the control's rectangle taking into account its height and width. The style animates the fore color of the circle when switching between true and false values (this can be turned off via $animateui).

The kCheckBoxRound button style supports a fill pattern or a transparent pattern set using $backpattern. If filled, the circle takes the colors from $forecolor and $secondaryforecolor for the checked (true) or unchecked (false) values, respectively. If transparent, the circle is not drawn and only the icon for the checked or unchecked state is displayed.

The round style uses $iconid and $secondaryiconid to control the icon displayed within the circle for the checked (true) or unchecked (false) values, respectively. Furthermore, when using themed SVG icons, you can use the $textcolor and $secondarytextcolor (for the checked and unchecked values, respectively) to set the color of the icons, which is white by default. For example, you could specify a green check icon when the checkbox is true and a red cross icon when the checkbox is false.

The default values for fore colors and text colors are inherited from the OS, for example, on macOS a round circle with nuanced blue is shown for true and a gray circle for false values alongside white icons.

Pushbuttons

Button Timer

You can now add a timer to a pushbutton using the **$timeout** property to delay the evClick event. (ST/HE/1779)

The **$timeout** property assigns a positive integer N to start a countdown timer that runs for N seconds, appending the time left to the button text; you can assign zero to stop the timer. When the timer expires, the button receives an evClick event with pTimeout set to kTrue to indicate the timer has finished. When $timeout is active, the text for the button updates once a second.

When sending evClick for the timeout, no evAfter is generated for the current field (this ensures the timeout event is received).

This is a runtime only property, and only assignable when the button has text.

Button Area

The $buttonmode property has been added to the Button Area window control allowing you to assign a user-defined or standard mode or action to the button area. (ST/WO/2701)

All the actions or modes available for the Pushbutton are supported for $buttonmode in the Button area (except for the color, linestyle, and pattern picker modes); this includes kBMuser, kBMok, and kBMcancel, as well as the actions for entering data into an Omnis datafile (these do not work for SQL databases). As with pushbuttons, kBMuser is the default mode, which allows you to define your own method to execute when the button area is clicked.

IDE Button Style

A new button style **kIDEButton** has been added to $buttonstyle for window class pushbuttons; this is intended to style buttons in the Omnis IDE, but you can use the style in your apps if you want. (ST/WO/2723)

The appearance for the kIDEButton buttonstyle is defined in a new section, **IDEbutton,** in the Appearance configuration file (appearance.json). You can override the border and text colors set in the theme, by setting them to a color other than kColorDefault, but note that a disabled button always uses the disabled text color.

Buttons styled with kIDEButton support hot tracking on both macOS and Windows, which can be controlled via the “hottracking” item in the “IDEbutton” section: 1 means buttons are hot on macOS, 2 means buttons are hot on Windows, 3 means buttons on both macOS and Windows are hot, and zero means hot tracking is not used on any platform.

Icon Color

Push Button controls now have the $iconcolor property which is the color when the icon is a themed SVG icon; the default is kColorDefault. Check boxes, Radio buttons, and Radio Group controls also have the $iconcolor property.

Themed SVG Icons

Window class controls now support SVG icons that have been “themed” using the Omnis Themer tool (themed SVG icons were introduced for JS components in previous versions). The ‘material’ icon set in Omnis contains themed SVG icons and is now available automatically when you edit a Window class to allow you to add the material icons to window components.

A themed SVG icon will use the color set in the **$textcolor** property of the window class control, so it matches the color of the text for the control. For Styled text, a themed SVG is drawn using the current text color for the text run.

A few external components have new properties to support themed SVG icons. The Multi-button, Round button and Tile external components now have the $textcolor property. The HTML icon link control now has the $::textcolor property. The color specified in these properties will be applied to a themed SVG icon.

Drag Icon background

As a consequence of support for themed SVG icons for window classes, drag icons now have a background by default on macOS, to prevent themed SVGs from becoming invisible, and make drag icons more cross-platform. You can turn off this behavior using the **dragIconBackground** item in the ‘macOS’ section of config.json (default is true, to show the icon background).

Dark and Light Modes

You can specify different SVG icons for Dark and Light modes; this enhancement is only intended for running desktop apps on Windows or macOS, since different system color modes do not apply to web and mobile apps running in a web browser.

Each icon set folder can now have two sub-folders, named **dark** and **light,** into which SVG icon files can be placed to support Dark and Light system color modes.

When you assign an icon to a control you only need to assign a single icon name or ID and the icon for Dark or Light mode will be chosen automatically from the appropriate sub-folder.

Paged Pane Buttons

The **$showpagebuttons** property has been added to the Paged pane control. When set to kTrue, a page indicator is shown at the bottom in the center of the paged pane to indicate which page is currently shown. The page indicator contains the same number

of dots as there are panes in the control. The end user can change the current pane by clicking on the page counter.

The $animatui property has also been added to Paged panes. When set to kTrue, the pages slide to the left or right when the page changes; when set to kFalse, the page pane will change instantly when the page button is clicked.

OBrowser

Back and Forward options

The **$contextmenuremovebackforward** property has been added to the OBrowser component, to hide or show the **Back** and **Forward** navigation items from the context menu for the component. (ST/EC/1667)

In addition, the **View Page Source...** item has been removed from the context menu, since this did nothing in the context of the embedded browser in OBrowser.

HTTP headers

The **$headerlist** property has been added to the OBrowser component, a runtime-only property, to set the HTTP headers tor the embedded browser in OBrowser. (ST/BR/387)

The $headerlist property takes a two-column list of HTTP headers to be added, or removed, for each URL request. Column 1 is the header name (without trailing colon). Column 2 is the header value, which you can set to #NULL to remove the header.

The list is applied in line order, so you can modify an existing header by removing it and then adding it. The list is applied to every request made after assigning $urlorcontrolname to a URL. The *referer header* cannot be changed using this method.

Headed List

Progress Bar

You can now display a Progress Bar in a column inside a Headed List box control, for example, to indicate a percentage value. (ST/GR/405)

To enable a progress bar to be displayed, there is a new text style **kEscBar** which can be used with the style() function (note **kEscBar** is not supported in the JavaScript client). When enabled, the progress bar sizes to the column width, so no other content is allowed in the column.

kEscBar draws a bar with 1 or 2 segments in a headed list column. kEscBar can take 3 or 5 parameters: the background color, segment 1 width (%), segment 1 color, then optionally segment 2 width (%), and segment 2 color.

For example, you can use the following in the calculation for a headed list column to draw a red segment of width iPercent % of the column width over a gray bar sized to the column width:

```
style(kEscBar,kGray,iPercent,kRed)
```

In this example, iPercent is a column value in the list.

Ellipses in Headed Lists

Ellipses are shown when there is not enough space to display all the content in a headed list cell (or in the text for a tree list node); this applies for headed lists with more than one column.

The **$disableellipsis** property has been added to the Headed List and Tree List window components to allow you to disable ellipses for individual list fields, if required.

This is in addition to the existing $clib.$prefs.$disableellipsis library preference which allows you to disable ellipses for all Headed Lists and Tree Lists.

Tooltips

The length of text in tooltips for Headed list boxes is now unlimited; in previous versions, the length was limited to 255 characters. (ST/WO/2695)

Although there is now no limit imposed on the tooltip length, in practice the absolute maximum would be 32000, and the longest reasonable size to use would be around 2k.

Resize Column

The $resizecolumn property has been added the Headed list. (ST/WO/2726)

The $resizecolumn property specifies the column that is resized appropriately when the width of the control changes, such as when using $edgefloat properties to resize the list when the window size changes. A value of zero means no column is resized, but the last column extends if necessary.

Complex Grid

Row Height

The **$getrowheight**([irow]) method has been added to Complex Grids which returns the height of the specified row. If iRow is not specified the height of the current row is returned. (ST/GR/401)

Resizing Rows

The ‘shiftRequiredToResizeAllRows’ configuration item has been added to the ‘complexgrid’ section of config.json to control the way Complex grid rows are resized when using the mouse and Shift key. (ST/GR/408)

If true (the default), dragging a row divider without pressing the Shift key resizes the single row above the row divider. To make all rows have the new row height, press the Shift key while dragging a row divider.

If false, the behavior is reversed. Press the Shift key while dragging a row divider in order to resize the single row above the row divider. Dragging a row divider without pressing the Shift key gives all rows the new row height.

Footer

You can now target the footer dynamically in a Complex Grid. (ST/GR/433)

To support this new feature the kGridOther constant has been renamed to kGridColumnFooter .

String Grid

The $cellleftpadding property has been added to the String grid control (and Data grid). This allows you to add content padding to the left of all cells in the grid. (ST/WO/2755)

Rounded Borders

You can now apply rounded borders to most Window class UI controls with the addition of the $borderradius property. (ST/WO/2700 & ST/WO/2702)

The **$borderradius** property has been added to many window controls to allow you to apply rounding to the control, including the following controls:

Single line entry	Multi-line entry
Masked entry	Token entry
Picture	List
Tree list	Headed list

Checkbox list	Dropdown list
Combo box	Scroll box
Complex grid	Data grid
String grid	Paged pane

The $borderradius property only applies when $effect is kBorderPlain, kBorderCtrlEdit or kBorderCtrlList.

In addition, the $borderradius property has been added to Pushbuttons, Radio buttons, and Check boxes when $buttonstyle is set to kUserButton (ST/WO/2667).

Styled Text

The $styledtext property has been added to the Tab pane component. When set to kTrue, the text for the tabs can now use styled text. (ST/WO/2654)

When $styledtext is kTrue, styled text can also be used for the $tabcaption and $alltabcaptions properties to display styled text in the tab captions.

In addition, styled text can be used in the $text property for push buttons, radio buttons and check boxes when $styledtext is kTrue.

Tree List

The $getnodetooltip method has been added to Tree List control. (ST/EM/234)

The $getnodetooltip(rNoderef) method is called (if $tooltip is empty) to get the tooltip to display when the mouse is over the specified node rNoderef. It returns either a styled text tooltip string, or an empty string (meaning use the default tooltip text).

To implement this method, right click on the method in the method tree, and override it.

Rounded Rectangle and Shape Field

The Rounded Rectangle background object and Shape Field now have the **$cornerradius** property, which is the pixel radius of the corners of the object (assign a value <= 0 to set this property to its default value). The maximum value of $cornerradius is 255.

In versions prior to Studio 11, rounded rectangles were not drawn with the same amount of rounding on Windows and macOS: when the value of $cornerradius is 0, this difference is maintained, for compatibility, and after converting a library to Studio 11, $cornerradius defaults to zero. When you set the value of $cornerradius to a positive integer (1-255), both platforms now use the same amount of rounding.

Tab Pane and Paged pane

The **$movetab** property has been added to the Tab pane, and the **$movepage** property has been added to the Paged pane. (ST/WO/2460)

The $movetab and $movepage properties allow you to move a tab or pane in design mode, which is useful when adding new tabs or panes and you need to reorder existing tabs or panes.

Picture Control

The $iconid and $iconcolor properties have been added to the Picture Control. (ST/WO/2730)

The $iconid property allows you to use an icon from an iconset in the control, such as an SVG icon, for example, to create a background image for a window. The $iconcolor property can be used to specify a color for a themed SVG icon. Setting either $dataname or $calculation takes precedence over $iconid.

Combo Box

You can now use $ctarget.$assign() to set the focus on a Combo box in a window toolbar. (ST/TB/336)

In previous versions you could not programmatically set the focus on a toolbar combo box, but now you can using $assign() and the following code:

```
Do $ctarget.$assign($cinst.$toolbars.tCombo.$objs.combo) ## or
Do $ctarget.$assign($itoolbars.tCombo.$objs.combo)
```

In addition, $ctarget.$assign() works when the window has no current field, but this does not work during $construct for any field. Therefore, there is another enhancement to allow *Queue set current field* to work for toolbar combo boxes (and on Windows, toolbar droplists). To do this, use the code

```
Queue set current field $cinst.$toolbars.t1.$objs.combo
```

which does work when executed during the window $construct.

Hyperlink

You can now add a separator line in the list of options in the Hyperlink control; this only applies when $vertical is kTrue.

To include a separator line, set the text in the $dataname list to a single - (hyphen) character; the line draws across the width of the control, inset by the left margin. The group id (in column 1) contains the color of the line; kColorDefault means use the IDE line color (as defined in appearance.json). For example:

```
    Do ihlkSubList.$add(kColorDefault,0,-)
```

Color Palette

The Color Palette now allows you to set the initial color, rather than using kDefaultcolor. (ST/WO/2754)

The $colorind property has a new mode 11 which takes a color passed as parameter 3; in this case parameter 1 is unused and should be passed as 0. If it finds the color in either of the 3 color palettes (256, 16 users, or 16 constants) it sets the current color. For example:

```
Do $cinst.$objs.colorpal.$colorind(0,11,kColorUser13), or
Do $cinst.$objs.colorpal.$colorind(0,11,rgb(255,0,0))
```

Window Toolbars on macOS

The **$toolbaroptions** property for window classes has two new constant values: kTBOptionmacOSExpanded to provide a macOS expanded style to allow a legacy toolbar appearance on macOS 11+, and kTBOptionmacOSCompressed to minimize the space between toolbar items for a macOS unified toolbar. (ST/TB/335 and ST/TB/334)

kTBOptionmacOSExpanded

When selected **kTBOptionmacOSExpanded** will place any toolbar displayed in the title bar of a window on macOS Big Sur or later below the title with the title centered as with previous versions of macOS. This setting is overridden by the global configuration file ‘useToolbarStyleExpanded item in the ‘macOS’ group in config.json (added in 10.2). When set to true all windows on macOS will use the expanded style. By default on macOS Big Sur and later a toolbar in the title bar will appear unified and be positioned next to the title.

kTBOptionmacOSCompressed

When selected **kTBOptionmacOSCompressed** will minimize the space between toolbar items for a macOS unified toolbar, i.e. where the toolbar title appears to the left of the toolbar items. The text label for items are also hidden. This option only has effect where the unified toolbar is supported (macOS Big Sur and later). For this option to be

active the kTBOptionmacOSExpanded or kTBOptionmacOSOmnisTopToolbar option cannot be set. Space can be added between toolbar items by using a blank kToolSpacer toolbar component.

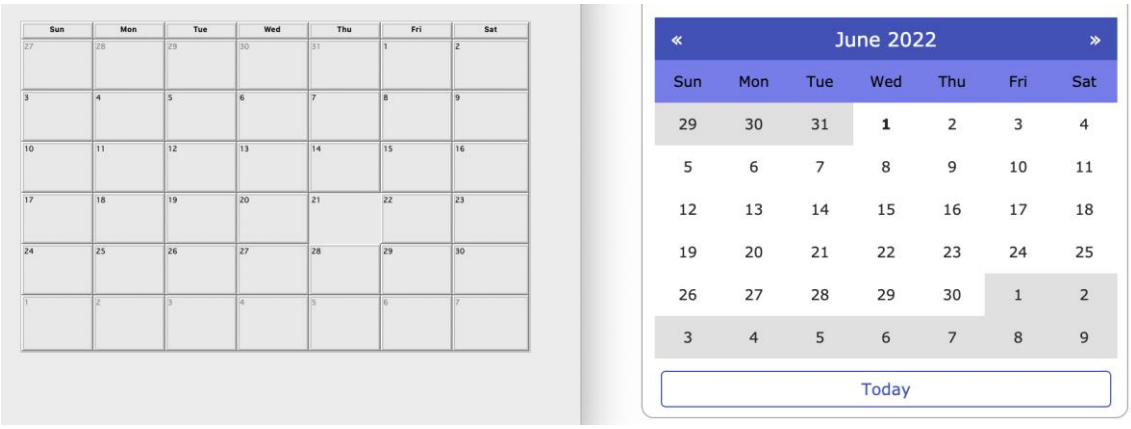
JavaScript Client Bridge

Due to issues sending messages to Omnis including Omnis dates inside lists/rows, any object members whose names begin "__" (double underscore) are now stripped out before sending to Omnis, e.g. in a 'sendControlEvent()'. (ST/EC/1774)

Calendar External Component

The UI in the Calendar external component has been enhanced to display dates using a modern interface (it is now more like the JS Calendar control, but only displays dates and not times). (ST/EC/1698)

The following screenshot shows the existing and the new UI for the Calendar external component.



The new **$uistyle** property must be set to **kCalUIstyleModern** to enable the new UI; the **kCalUIstyleClassic** setting maintains the existing drawing style (the default). The following properties are only available when $uistyle is set to kCalUIstyleModern:

Property	Description
$navcolor	The color used for the navigation bar
$navfont	The font used for the navigation bar
$navfontsize	The fontsize used for the navigation bar
$navtextcolor	The color of the text in the navigation section
$showmonthnav	If true, if the navigator bar is shown
$weeknumbercolor	The color for the background of week numbers if shown
$weeknumbertextcolor	The text color for week numbers if shown
$showweeknumber	If true, the week number column is shown
$setdayicon	Sets the day icon

The following existing properties are ignored if $uistyle is set to kCalUIstyleModern:

$daymode
$currdaymode
$headingmode
$otherdaymode

Navigation bar

When the **$showmonthnav** property is set to kTrue the navigation bar is shown. Clicking on the right or left arrow in the navigation bar will change the month view.

If you click on the month displayed in the navigation bar the month selector is shown in the main calendar, and likewise selecting the year in the navigation bar the year selector is shown in the main calendar. Selecting a cell from any selector returns the calendar to the previous selector and eventually to the default mode.

If **$allowchange** is set to false, the left and right navigation buttons are removed and various navigation selectors unavailable.

If **$showweeknumbers** is set to kTrue, week numbers are displayed down the left side of the calendar; the color of the week number text and background is controlled using the $weeknumbertextcolor and $weeknumbercolor properties, respectively.

Window Programming

Toast Messages

The $toastnotificationclicked() method has been added to startup tasks which you can use to determine when a toast notification message has been clicked. (ST/NT/794)

The $showtoast() method has a new optional parameter pContext, which can be passed into $toastnotificationclicked when the content of the toast is clicked, and after querying pContext your code could take some specific action, such as opening a window.

If you return kTrue from $toastnotificationclicked, the toast is closed immediately. $toastnotificationclicked is not called if clicking the toast close box.

Window Minimum Size

The **$minwidth** and **$minheight** properties have been added to the window class to allow you to set a minimum width and height for a window. This can be useful if you want to stop the end user making a window containing many floating fields too small, to preserve its layout, for example. (ST/WC/569)

$minwidth and $minheight are the minimum values to which the $width or $height properties of a window can be set, either programmatically or by the user resizing the window. A value of zero (the default) means that no minimum is specified.

Window Animations

Certain properties of window instances can now be animated using the $beginanimations() and $commitanimations() methods, including $alpha, $left, $top, $width, and $height. So for example, when closing a window you could fade it out by setting $alpha under animation before closing it, or similarly you could enlarge a window under animation as you open it to create more impact.

Simple Style Windows

The $growbox property is now assignable for simple style windows ($style = kSimple), as this controls whether the window has a sizing border. This will allow you to make simple style windows look more alike on macOS and wWindows. (ST/WC/580)

For a window with $style set to kSimple, irrespective of the setting of $growbox, you can always resize the design window. On Windows, there is a wider sizing border in design mode, even if $growbox is kFalse. When you open the window, the open window is only resizable if $growbox is true, and on Windows it only has a wider sizing border in this case.

The net effect is that for kSimple style windows with $growbox kFalse, there is no wide sizing border on either platform for the open window, making them appear more alike.

Window Title Colors on macOS

The colors for window title bars on macOS can be set using new settings in the appearance.json file. (ST/HE/1778)

macOS now uses the **coloractivecaption** and **colorinactivecaption** items in the 'system' section of appearance.json to provide the colors for window title bars. If either of these is set to kColorDefault, the system default colors are used.

In addition, there is a new entry **macoscaptiontextappearance** in the 'system' section of appearance.json that specifies the text color for captions (the window title); this is an integer: 0 system default, 1 dark text, 2 light text.

Docking Areas & $screen property on macOS

On macOS, the $screen property (a property of $toplevelhwnd) now excludes a visible macOS dock from its coordinates. (ST/HE/1625)

There is a new item **excludeDockFrom$screen** in the 'macos' section of config.json, which is true by default. A reference to $screen can be obtained using the following code:

```
Set reference lItem to $cinst.$toplevelhwnd.$ref
Set reference lItem to lItem.$screen.$ref
```

Bitmap Image Conversion

On macOS, the pictconvto() function now supports conversion from all supported bitmap data formats, i.e. TIFF, BMP, JPEG, GIF, etc. (ST/WO/2669)

Where a source data format is empty, an attempt will be made to convert from any supported bitmap image representation, e.g. TIFF, BMP, JPEG, GIF, etc. The data must be in raw format with no Omnis header.

Masked Entry Fields

The $formatstring property in Masked Entry Fields has been enhanced to allow you to display floating Decimal Point numbers. (ST/FU/825)

When defining an input mask in $formatstring, you can now use D (or d) in place of . in a number format section, forcing Omnis to add at most the decimal places specified in the format, with no trailing zeroes and no trailing decimal point.

For example, you can use #,##0D000 to format numbers to include a thousands separator and up to 3 decimal places. The formatted result will not contain any trailing zeroes or a trailing decimal point.

The popup for defining numeric formats in $formatstring now includes a button for D.

JSON Components

SVG Icons

The icon for a JSON defined control can now be SVG and it can be themed. (ST/JS/2787)

The SVG image file must be placed in the folder with control.json, and the JSON control will use it as the control icon in the Component Store.

SQL Programming

Debugging Slow Queries

A new property **$trackslowqueries** has been added to the Omnis DAMs to allow you to track and debug slow queries. (ST/*A/162)

The value of $trackslowqueries represents the number of seconds that an EXECUTE or FETCH from the database has to reach before being considered slow. The default is 0 which means the query tracking is off.

For example, if the value of $trackslowqueries is 2 and a query takes 3347 milliseconds to finish, the query will be reported to the trace log alongside its execution time. If $debuglevel is set to 2 and a $debugfile is set, the slow query will be reported in both trace log and file specified in $debugfile.

Part of the SQL executed will be included in the message, up to 80 characters, in order to keep the logs clean; this should be enough to identify the query in the code.

updatenames() List Method

Three new parameters have been added to the $updatenames() method for a list with a table instance that allow the Where clause to be omitted and the updated columns to be determined from the data that has changed. (ST/NT/789)

The definition for $updatenames() is now:

- **list.$updatenames**([cOldrowName] [,cRowName='', **bExcludeWhere**=kFalse, **wOldrow**=#NULL, **wRow**=#NULL])

Pass bExcludeWhere as kTrue to exclude the Where clause. This is false by default. If wOldRow is supplied, then wRow can also be supplied, or if not, the current line of the list will be used (so if wRow is omitted and there is no current line, $updatenames fails and returns #NULL).

When wOldRow is supplied, in addition to the usual behavior of omitting columns marked as $excludefromupdate, $updatenames also excludes columns where the value in wOldRow equals the value in wRow (or the current list line if wRow is not supplied).

Omnis VCS

VCS API

Some of the functions of the Omnis VCS have been exposed to allow you to interact with the VCS or the contents of a project programmatically. (ST/VC/725)

You can make API calls to the Omnis VCS by calling:

$root.$modes.**$dotoolmethod(**kEnvToolVcs,**'vcs_method_name'**[,parameters,..]),

The first parameter is always kEnvToolVcs to specify a VCS method, followed by the VCS method name and the appropriate parameters.

Tokens

Each API call requires the use of a *token,* which is a unique string generated by the VCS when a successful logon occurs. This token is an essential parameter to all the calls (apart from $x_logonVCS) as it is the mechanism that the VCS uses to verify the validity of the API call. By default a token will last for 60 minutes, but you can extend the token lifespan to up to 8 hours. When the token time has expired, you will be logged off automatically and will need to logon again.

Logon

The $x_logonVCS method allows you to logon to the Omnis VCS using an existing SQL session, returning a token which must be passed by the other methods.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_logonVCS',cHOSTNAME,cUSERNAME,cPASSWORD,
   nTokenTime,cToken,cErrors) Returns bStatus
```

cHOSTNAME, cUSERNAME, cPASSWORD are character strings used to identify the session to log onto. The session must have been previously set up via the SQL Browser Session Manager. cHOSTNAME is the name of the VCS session previously set up and cUSERNAME and cPASSWORD are the credentials that are setup inside the VCS (not the database credentials).

nTokenTime is a value to determine (in minutes) how long the token will remain valid for. If 0 is passed, a default value of 60 minutes is applied; the token time can be up to a maximum value of 480 (8 hours).

If the logon is successful, **bStatus** will return kTrue and **cToken** will contain a token generated by the VCS which must be used to authenticate subsequent API requests.

As with all the methods (except Logoff), **cErrors** will contain any errors.

Logoff

The $x_logoffVCS method logs out of the current VCS session.

```
Do $root.$modes.$dotoolmethod(kEnvToolVcs,'$x_logoffVCS',cToken)
```

You need to pass cToken to logoff.

Get Token Info

The $x_getTokenInfo method returns information about the token and session.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_getTokenInfo',cToken,rRow,cErrors) Returns bStatus
```

If successful, **rRow** lists the Token, Token Expiry Time, the session name you are logged on to, VCS username you are logged on with, the token timeout in minutes and the Logon time.

List Projects

The $x_listProjects method returns a list of projects.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_listProjects',lLibList,cToken,cErrors) Returns bStatus
```

If successful, lLibList will contain the list of projects that are available in the VCS repository.

Class Status

The $x_classStatus method returns the status of a class, its checked out status, who checked it out, and so on.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_classStatus',refClassRef,rRow,cToken,cErrors) Returns
   bStatus
```

refClassRef is a reference to a class in your local library. If the call is successful, the row variable **rRow** will be populated with the name of the class, its checked out status, who checked it out and when, the date of the last revision and who checked it in, as well as the current revision number.

Checked Out Classes

The $x_checkedOutClasses method returns a list of checked out classes.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_checkedOutClasses',cUserName,[cLibName],lList,cToken,cErrors
   ) Returns bStatus
```

cUserName is a character string for the VCS user name. The optional parameter **cLibName** filters the list of checked out classes to the supplied library name. If the call is successful, **lList** will be a list of classes containing project name, user name, class type, class name, checked out date, check out notes and the original library name.

Is Class Current

The $x_isClassCurrent method tells you if a local class is up to date or not.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_isClassCurrent',refClassRef,cClassStatus,cToken,cErrors)
   Returns bStatus
```

refClassRef is a reference to a class in your local library. If the call is successful, **cClassStatus** will contain either 0 or 1: if 0, the class is up to date with the VCS, or if 1, the VCS version is newer than the local copy.

Check Out

The $x_checkOut method allows you to check out or copy out a class.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_checkOut',refClassRef,refLibRef,cToken,bCheckOrCopy,cErrors)
   Returns bStatus
```

refClassRef is a reference to a class in your local library and **refLibRef** is a reference to the library you are checking the class out to. **bCheckOrCopy** is a boolean allowing you to either check out (kTrue) or copy out the class (kFalse).

Check In

The $x_checkIn method allows you to check in a class.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_checkIn',refClassRef,refLibRef,cToken,cErrors) Returns
   bStatus
```

refClassRef is a reference to a class in your local library and **refLibRef** is a reference to the library you are checking in from. The project must already exist in the VCS as it is not currently possible to create a new project using the API.

Label Project

The $x_labelProject method allows you to label a project.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_labelProject',cProject,cLabel,cToken,bOverwrite,cErrors)
   Returns bStatus
```

cProject is the name of the project, **cLabel** is the label and **bOverwrite** indicates whether to overwrite an existing label of the same name.

Build Project

The $x_buildProject method allows you to build a project to a specified folder.

```
Do $root.$modes.$dotoolmethod(
   kEnvToolVcs,'$x_buildProject',cProject,cBuildPath,cLabel,
   bLocked,bOverwrite,bLowercase,cToken,cErrors) Returns bStatus
```

cProject is the name of the project, **cBuildPath** is the directory to build into, **cLabel** is the label to use, **bLocked** indicates whether to build a locked library, **bOverwrite** identifies whether to overwrite an existing library, **bLowercase** builds the file name in lowercase.

VCS Auto Login

There is a new option in VCS sessions to allow you to logon automatically at startup. (ST/VC/695)

For VCS sessions, there is a new 'VCS' tab that allows you to enter the VCS username and password which will be used to log onto the VCS automatically when you start Omnis. If this is an existing session, there is a button which allows you to verify the username/password are correct for the session. Then on the Session Definition tab you need to enable the 'Logon at Startup' option.

VCS Check in/out Options

The VCS Check in/out options can now be shown on one screen, without tabs. (ST/VC/767)

The **VCS Options** has a new option **Show All Options Without Tabs** (included on the Check in & Check out tabs) to allow you to show all Check in/out options in one screen (by hiding the tabs) when classes are checked in/out.

Initial Library Check in

The VCS build properties are now set automatically when a library is checked in for the first time, setting the classes to read-only. (ST/VC/792)

The is a new option **Make library read-only when checking in for the first time** on the **Check In** tab under the **VCS options** to control this behavior; the default is enabled, therefore set this to false to restore behavior in previous versions.

List Programming

List Methods

The LIST.$count() method has been added to list variables to return the count of lines in a list. (ST/FU/813)

The method LIST.$count([bSelectedLinesOnly=kFalse]) returns the count of lines in a list, optionally passing bSelectedLinesOnly as kTrue to only count selected lines. This enhancement extends LIST.$count() to whole lists, since the method has been available for list columns in previous versions.

Report Programming

Report Fields

The maximum number of report fields allowed in a report class has been increased to **8191** from 3000. (ST/RC/1383)

You are advised to split large reports (containing a large number of report fields) into a number of sub-reports, and print them to either the Printer or PDF using the *Begin* and *End print job* commands.

Page Preview Zoom Factor

You can now specify the initial zoom level for the Page Preview report destination. (ST/RC/1347)

The '/zoom=n' parameter can now be included directly after the 'Title' parameter when executing the *Send to page preview* command, or setting the $windowprefs preference.

- **Send to page preview** syntax is now
  Send to page preview ([Do not wait for user][,Hide until complete])
  title[/zoom=n][/left/top/width/height/cen/max/stk]
- **$prefs.$windowprefs** setting is
  Title[/ZOOM=n][/left/top/width/height/CEN/MAX/STK]) sets the position and initial zoom factor for preview windows

The string /zoom=n is optional, where n is an integer zoom factor, while a zero value means zoom fit. If omitted the command or preference behaves as in previous versions, i.e. the Preview window is zoomed to fit the screen.

A value for n of -1 to -7 means use the standard zoom factor indexed using -n (1 to 7); this corresponds to the 7 standard zoom factors for the window, in ascending order.

A positive value means use the standard zoom factor closest to, but not exceeding, n. So you can pass in 175 for example to have an initial zoom factor of 175%.

Report Data Grid Column Parameters

Column calculation properties (in $::calculation) for the Report Data Grid component are now tokenized so that they work with the current function parameter separator. (ST/EC/1649)

For compatibility with previous versions, the component still works with the $::calculation and $columnheader properties stored as character strings, provided that the function parameter separators match those currently in use. When you re-enter one of these properties (select the property in the Property Manager and press Return) the property changes so that it is stored as a tokenized calculation, which will then work with any function parameter separator.

An alternative way to convert a library is to export to JSON and re-import, and in this case Omnis tokenizes the calculations on import. For import, Omnis will accept the report list calculations as either character strings or calculations; import always results in tokenized calculations being stored. Accepting both forms means that import is compatible with JSON exported in previous versions.

Report Preview URL Prefix

The **omnisPreviewURLPrefix** item has been added to the 'defaults' section of config.json to allow you to set the report preview URL prefix for the $linkaddress and $address properties for report class Entry fields and HTML Link objects. The item defaults to 'omnis:' if empty. (ST/RC/1340)

Omnis Programming

User Constants

User constants allow you to define constants in a new **User Constants** class for use in your methods and expressions. A user constant is a named value, where the value cannot be changed during execution. Generally speaking, user constants can be used anywhere in Omnis code and expressions, although there are exceptions, because they cannot be used anywhere that would attempt to modify them, for example:

- As the result of a Calculate command

- As the Returns component of commands such as Do
- As the dataname of a variable

To define user constants, you add their names and values to a new class type, the **User Constants** class. The types and therefore values are restricted to Character, Integer, Number and Boolean. You can have multiple user constants classes, each of which defines a number of user constants and their values.

Internally, user constants are handled as a special type of file class, meaning that the same naming rules as those for file classes apply, i.e. $clib.$prefs.$uniquefieldnames, $clib. $prefs.$sensitivefieldnames and $clib.$prefs.$sensitivefilenames all apply (note that this means $sensitivefilenames and $uniquefieldnames are now included in JavaScript-only development editions of Omnis Studio). For naming and tokenisation purposes, user constant names are essentially file class variable names.

Also, user constant classes are always treated as memory only, and file class commands such as Clear all files, and Set memory-only files have no affect on user constant class CRBs.

When exporting a library using the JSON export, user constants classes are included, using a similar syntax to that used for file classes.

Creating User Constants

The **New Class** hyperlink in the Studio Browser and **New Class** hierarchical context menu have a **User constants** command, to create a new User Constants class. There is also a class filter that controls whether user constant classes are visible.

There is a new editor for user constants classes, where you can define the name, type, subtype, value and description of user constants. User constants can be named however you like, including the prefixes "k" and "ev" which are used for built-in constants and events.

If you try to delete a user constant, the editor will check the current library to see if the constant is in use, and warn you about this.

The **Catalog** has a new tab, named **User Constants.** This has similar behavior to the Variables tab.

Method Editor and Code Assistant

User constants have a new syntax color and style, in the IDEmethodSyntax group of appearance.json: userconstantcolor and userconstantstyle.

The code assistant default sort order includes user constants at the start of the list, sorted with instance variables, etc.

Also, the option click menu, opened when you right click on a user constant, is a subset of that which applies when you right click on a file class variable.

The Method Editor and other editors in the IDE have validations to prevent user constants from being used where their value could be changed. Similarly, debugger variable windows do not allow user constants to be modified. However, it is impossible for the IDE to detect every such situation e.g. due to expressions generated at runtime using square bracket notation, so in addition, as a fallback, the low-level code managing the CRB also checks for attempts to modify a user constant, and generates a runtime error if something attempts to do this.

Notation

There is a new notation group in $clib, named $userconstants, supporting similar notation to $files. However, user constants classes do not have $conns, $datahead or $indexes members, and user constant objects only have the properties $desc, $ident, $name, $objinitval, $objtype, $objsubtype, $objsublen and $userinfo. $objinitval contains the value of the constant.

Using the class notation for a user constants class is the only way you can programmatically modify the value of a user constant.

The $classtype value for a user constants class is kUserConstants.

Adding Method Lines

You can now use $addbefore() and $addafter() with the $methodlines class method property. (ST/NT/800)

- **$addbefore**(rItem,cText)
  adds a new line with content cText before the line specified by rItem (rItem can be either a 1-based integer line number, or an item reference to a line in the method)
- **$addafter**(rItem,cText)
  adds a new line with content cText after the line specified by rItem (rItem can be either a 1-based integer line number, or an item reference to a line in the method)

For example:

```
Do $cclass.$methods.$remove($cclass.$methods.Test)
Do $cclass.$methods.$add("Test")
Do $cclass.$methods.Test.$methodlines.$add("# aaa")
Do $cclass.$methods.Test.$methodlines.$add("# ccc")
Do $cclass.$methods.Test.$methodlines.$add("# eee")
Do $cclass.$methods.Test.$methodlines.$addbefore(2,"# bbb")
Do $cclass.$methods.Test.$methodlines.$addafter(3,"# ddd")
Do
   $cclass.$methods.Test.$methodlines.$addbefore($cclass.$methods.Test.$methodl
   ines.1,"# New line 1")
Do
   $cclass.$methods.Test.$methodlines.$addafter($cclass.$methods.Test.$methodli
   nes.2,"# New line 3")
```

Max Chain Depth

The **maxChainDepth** item has been added to the 'defaults' section of config.json which allows you to configure the maximum number of field or item references that Omnis will chain through in order to reach the referenced variable. (ST/VR/328)

The default or minimum is 20, and in all but exceptional cases, you should leave this item set to 20. You can change it if you have a heavily recursive method that uses field reference parameters. Since the minimum value is 20, setting this to any value less than 20 results in Omnis using the value 20.

The debugger field menu still only chains through up to 20 references.

Initial Parameter Values

The way parameter variables are initialized has changed: any parameters that are omitted when you call a method are now initialized using their initial value. This is the default behavior for new libraries. (ST/NT/781)

A new library preference, $clib.$prefs.$useoldparameterpassing has been added to control this behavior. If true, an empty parameter that is not the last parameter is initialized to empty or zero, rather than its initial value in the called method parameter definition (this does not apply to client executed client methods in the JavaScript client). The new library preference defaults to false in new libraries, and true in converted libraries to maintain backwards compatibility.

Item Group Methods

The group methods $makelist, $appendlist etc now allow you to list all objects in the item group ignoring any containers. (ST/FU/822)

If the first argument is now the constant kRecursive, the item group methods $makelist, $appendlist, $insertlist, $count for window class and instance $objs and $bobjs groups, and remote form class $objs groups, now ignore containers and adds all objects to the

returned list. The bRecursive argument has also been added to $sendall(), that applies to window class and instance $objs and $bobjs groups, and remote form class $objs groups.

Collecting Performance Data

Performance data collection no longer updates the $...executiontime properties (min, max, total) when the method is being called recursively, however it still updates $callcount for recursive calls. (ST/PF/1281)

Notation Error Checks

The 'stricterNotationErrorChecks' configuration item is now set to true by default. (ST/NT/782)

Error Reporting for External Components

A new item "allExternalComponentErrorsAreFatal" has been added to the "defaults" section of the Omnis configuration file (config.json) to manage whether or not #ERRCODE and #ERRTEXT are reported by external components. (ST/PF/1338)

When allExternalComponentErrorsAreFatal is true (the default), and an external component sets #ERRCODE and #ERRTEXT, the error always generates a runtime error, entering the debugger in the development version of Omnis.

Web Services

HTTP Methods

The $sethttpstatus method now defaults to 200. (ST/WS/330)

RESTful APIs now no longer require $sethttpstatus to be called; if it is not called, the status defaults to 200 OK.

Escaping String Parameters

The 'Escape query string parameters' option has been added to the RESTful panel, allowing you to control whether or not string parameters are URI escaped. (ST/WS/332)

The 'Escape query string parameters' option defaults to true (replicating the behavior in previous versions), meaning that query string parameters are URI escaped. When turned off, the query parameters are not URI escaped, allowing you to perform any character encoding conversion yourself. For example, if you receive UTF-8 data instead of ASCII, you could turn this option off and escape the text using the ow3.escapeuritext() function.

Web and Email Communications

OW3 LDAP Worker

A new **LDAP Worker** has been added to the OW3 group of worker objects, named LDAPClientWorker. Lightweight Directory Access Protocol (LDAP) allows "the sharing of information about users, systems, networks, services, and applications throughout the network" (Wikipedia).

To use the LDAP Worker, the $init method has the following definition:

- $init(cURI,cUser,cPassword) Initialise the object so it is ready to access the specified URI using LDAP.Returns true if successful
  cURI: The URI of the server, optionally including the URI scheme (ldap or ldaps)

e.g. ldap://ldap.myserver.com. If you omit the URI scheme e.g. ldap.myserver.com, the URI scheme defaults to ldap
cUser: The user name to be used to log on to the LDAP server
cPassword: The password to be used to log on to the LDAP server

After calling $init(), you can call $run() or $start(), as the details of what is to be retrieved from the server are passed in the URL. The standard RFC4516 defines the syntax of LDAP URLs (https://docs.ldap.com/specs/rfc4516.txt).

When the query completes, the worker calls $completed in the usual way for OW3 workers. The row passed to $completed has 4 columns:

errorCode: Zero for success, otherwise an error code

errorInfo: The description of the error

log: If you enabled logging for the OW3 worker, this contains the log

rawData: If successful, the result of the LDAP query. The developer is currently responsible for parsing this.

OW3 Python Worker

A new **Python Worker** has been added to the OW3 group of worker objects. (ST/EC/1749)

The Python worker works exactly like the JavaScript worker, including support for HTTP/2, with a few exceptions, as follows.

The $init method has only one parameter instead of two, since Omnis does not support the remote debugger capabilities in Python.

When passing rows to the Python worker, a dictionary object is created in your Python module. When passing lists, a list object is created.

Installation

The Python executable is not provided with Omnis, so you will have to install it manually alongside pip (the package manager) on your chosen platform. The Python worker will work with python3 only and ideally you should use at least Python 3.6. Get the latest download and installation notes for different platforms from: https://www.python.org/

On Linux and macOS, Omnis expects the binary to be in /usr/bin/python3 as well as on the PATH. On Windows, Omnis expects the installation to also be in the PATH as it relies on loading the python3.dll to get the directory where Python is installed and use the python.exe within. Currently, you cannot specify a path to a different python executable to use.

In addition, flask, psutils and requests are required; these are listed in a file called requirements.txt which is in the pyworker folder in the Omnis read/write directory. You can either install them manually or by doing pip/pip3 install -r path/to/file/requirements.txt

In order to create a Python module, create a new folder inside the pyworker folder in the Omnis read/write directory, and include a main.py file which Omnis will load at runtime. When calling your module, use the folder name of your module and a function within your main.py. You can import omnis_calls in your main.py and use sendResponse or sendError if required, or you can simply return a message or some data, or raise an Exception if an error occurred (in which case it should automatically return a $methoderror or $methodreturn).

HTTP/2 support for OW3 Workers

The OW3 Workers have been enhanced to support HTTP/2 which is more secure as it uses binary protocols instead of plaintext, and is generally faster and more efficient for web communication. (ST/EC/1717)

The nghttp2 open source library has been added to accommodate HTTP/2 support, and various libraries have been updated including: zlib, mbedTLS, libssh2, and libcurl; if your application uses OW3 your product licensing should include the appropriate third-party licensing.

In addition, KOW3cryptoTypeBlowfish for the OW3 CRYPTOWorker has been deprecated (and will be removed in a future release) since there are more secure encryption algorithms available.

OW3 Worker Methods

The OW3.$parserfc3339() static method has been added to return an Omnis date-time value from a RFC3339 formatted time. (ST/VR/330)

- **$parserfc3339**(cRfc3339[,bUTC=kTrue,&iOffset,&cErrorText])
  parses a date and time value conforming to RFC3339 and returns an Omnis date-time value and optionally the time zone offset in minutes.
  Returns #NULL if the string cannot be parsed.

The parameters are:

Parameter	Description
cRfc3339	a date and time string conforming to RFC3339
bUTC	If true, the returned date-time value is in UTC rather than the local time zone of the RFC3339 date-time value
iOffset	If the RFC3339 date and time string is parsed successfully this receives the time zone offset in minutes
cErrorText	If supplied, receives text describing the error that caused $parserfc3339 to return #NULL

OW3 OAUTH2 Worker

Grant Types

The OW3 OAUTH2 Worker Object now supports multiple grant types: authorization_code, password, and client_credentials (as per RFC 6749 sections 4.1, 4.3 and 4.4). (ST/EC/1658)

The new **$granttype** property takes one of the following grant types:

- **kOW3OAUTH2grantAuthorizationCode** (the default)
  the Authorization Code grant type (behaves as previous versions)
- **kOW3OAUTH2grantPassword**
  the Password grant type requires the new $username and $password properties to be specified
- **kOW3OAUTH2grantClientCredentials**
  the Client Credentials grant type requires $clientid and $clientsecret

The $granttype property is set to kOW3OAUTH2grantAuthorizationCode by default which corresponds to behavior in previous versions, so your existing code should run as before.

When $granttype is set to kOW3OAUTH2grantPassword, the new **$password** and **$username** properties are used to retrieve an authorization token. However this is deemed to be insecure, when compared to more secure methods, and should not be used (unless a legacy system requires it).

When $granttype is set to kOW3OAUTH2grantClientCredentials, the properties **$clientid** and **$clientsecret** are used to obtain the authorization token. Note that when using this grant type, the OAUTH2 server may not return a refresh token (as per RFC 6749 section 4.4.3).

OW3 HTTP Worker

The OW3 HTTP Worker Object now converts a POST payload to JSON automatically. (ST/JA/006)

vContent of a HTTP POST request can be kOW3httpMultiPartFormData or a Binary/Character/List/Row variable. The new behavior allows you to pass a raw List or Row which the worker will subsequently transform into JSON before executing the request. If a row is passed and it contains only one column, which is a path to a file that exists, the contents of the file will be used. If the file does not exist, the contents of the row will be converted to JSON and sent with the request.

Existing code bases which pass JSON in a binary variable are not affected.

The lists can contain sub-lists as this change supports both JSON arrays of arrays, and arrays of objects.

OW3 FTP Worker

There is a new action **kOW3ftpActionMove** in the FTP Worker Object to move or rename a file on the FTP server. (ST/EC/1659)

cServerPath in $init is the pathname of the file or directory to be moved. vParam is the new server path name. The action works with FTP, FTPS and SFTP (the latter uses a different command), and can be used to rename a file, or move it to a new location.

OW3 JavaScript Worker

Example app

There is a new example app called **JS Worker** under the **Samples** section of the **Hub** in the **Studio Browser** (note there is a New option to display the new examples only).

Security

Security for the JS Worker has been improved, including encrypting all traffic sent between the Omnis & Node JS processes. (ST/JS/3246)

Auto Loading modules

The JS Worker will now pick up any modules you have added automatically, if they are placed in the jsworker folder. (ST/EC/1754)

Any modules added in their own folder (with a package.json or index.js) inside the jsworker folder are now picked up automatically by the JS Worker. You can continue using the hard-coded moduleMap method, as in previous versions.

Error Handling

The JS Worker now handles default error messages in the component, rather than sending with the response. (ST/JS/3247)

The status codes of the returned errors when a method/module is not found have changed from 400 for both method/module, to 460 for module not found, and 461 for method not found.

Caller tag

An optional vTag column has been added to the row parameter for $callmethod in the JS Worker Object. (ST/EC/1668)

If supplied, some data can be passed to $methoderror or $methodreturn in the column __tag of the row parameter. This can be used, for example, to identify the caller when the worker object is shared by several instances.

OW3 IMAP Worker

A new action kOW3imapActionSelect has been added to the IMAP Worker which executes an IMAP SELECT on the mailbox given to $init. (ST/EC/1772)

As part of the SELECT, IMAP returns the number of emails in the mailbox, which are returned to the $completed method in resultList in column EXISTS. Note that an IMAP SELECT will cause the current mailbox to be changed, so you may prefer to execute this action on a different connection.

Menu Classes

Menu Instances

$menuinst for a Cascading menu object and a Popup menu window object now appears in the Notation Inspector rather than the Property Manager. (ST/*A/142)

This allows you to select $menuinst and see the properties of the instance, and also drill down further to see $objs, etc.

Note that $menuinst only appears in the Notation Inspector when the parent object has an associated instance.

Menu Shortcuts (macOS)

A new item **useFnInMenuShortcuts** has been added to the "macOS" section of the config.json file to control how Function and Command keys on macOS are interpreted. When set to true (the default), the Function+number menu short cuts display as Fn, or if false they display as <CmdKeySymbol>+n. (ST/MC/264)

Menu Line Icon Colors

Menu classes now support the $iconcolor and $defaulticoncolor properties to control the color of icons when using themed SVG icons. This enhancement also applies to Toolbar classes and Tree list window controls.

The **$iconcolor** property for a menu line (or toolbar button) sets the icon color when using a themed SVG icon. The **$defaulticoncolor** property for a menu class (or toolbar) sets the icon color when using themed SVG icons *and the $iconcolor property of the item is kColorDefault*. If $defaulticoncolor is also kColorDefault, then themed icons use the text color.

Object Oriented Programming

Window Status Bar

A window subclass can now inherit window status bar panes from a superclass. (ST/WC/583)

In design mode, when you right click on the status bar, there is an additional **Inherit** context menu option which allows you to inherit the status bar panes from a superclass (the option allows you to toggle the panes on or off). When inherited, you cannot change any of the pane or bar properties in design mode, and these properties are displayed in the Property Manager using the inherited colour.

JSON export has been modified to export the status bar inherited flag. The original pane definitions still remain in the class after inheriting, so these are restored if you turn off the inherit option.

Subclass Editors

Class editors for subclasses now update immediately when the superclass has been changed, so you no longer have to close and re-open a subclass to see the changes made to the superclass (as in previous versions). This enhancement applies to the class editors for Remote form, Window, Menu and Toolbar classes.

Functions

Example apps

There are new example apps called **OIMAGE Functions** and **JS TOTP Passwords** under the **Samples** section of the **Hub** in the **Studio Browser** (note there is a New option to display the new examples only).

The following functions have been added or updated:

binfrombase32()

Function group	Execute on client	Platform(s)
Binary Field	NO	All

Syntax

binfrombase32(*vData*)

Description

Decodes the binary or character vData from BASE32 and returns the resulting binary data. Returns #NULL if vData is not valid BASE32 or an error occurs.

bintobase32()

Function group	Execute on client	Platform(s)
Binary Field	NO	All

Syntax

bintobase32(*vData*)

Description

Encodes vData as BASE32 and returns the result. vData can be either binary or character. If vData is character, Omnis converts it to UTF-8 before encoding it as BASE32. Returns #NULL if an error occurs.

charcount()

Function group	Execute on client	Platform(s)
String	YES	All

Syntax

charcount(*string*,*char*)

Description

Returns the number of occurrences of the character *char* in the *string*.

Example

```
Calculate lString as 'Omnis Studio is great'

Calculate lCount as charcount(lString,'i')
# returns 3


If charcount(lString,'i')>2
  # returned True
End If
```

complementarycolor()

Function group	Execute on client	Platform(s)
General	NO	All

Syntax

complementarycolor(*color*)

Description

Returns the complementary color given the passed *color* value. A complementary color is a color on the opposite side of the color wheel.

contains()

Function group	Execute on client	Platform(s)
String	YES	All

Syntax

contains(*string*,*substring*[,*ignorecase*=kfalse])

Description

Returns true if the *string* contains the non-empty string *substring*. If *ignorecase* is kTrue, the function uses case-insensitive comparison.

See also startswith() and endswith().

Example

```
Calculate lString as 'Build better software faster'

If contains(lString,'software')
  # returned True
End If

Calculate lBoolean as contains(lString,'SOFTWARE',kTrue)
# lBoolean = True
```

endswith()

Function group	Execute on client	Platform(s)
String	YES	All

Syntax

endswith(*string*,*end*[,*ignorecase*=kfalse])

Description

Returns true if the *string* ends with the non-empty string *end*. If *ignorecase* is kTrue, the function uses case-insensitive comparison.

See also contains() and startswith().

Example

```
Calculate lString as 'One Code'

If endswith(lString,'Code')
  # returned True
End If

Calculate lBoolean as endswith(lString,'CODE',kTrue)
# lBoolean = True
```

FileOps.$getfileinfo()

The FileOps.$getfileinfo function can now return the size of a directory. (ST/FU/796)

FileOps.$getfileinfo has a new constant kFileOpsInfoCalcDirectorySize which calculates the size of the passed directory path. The "size" is returned in bytes in the CalcDirectorySize column.

FileOps.$putfilename()

When running on Windows, the dialog opened by FileOps.$putfilename() now prompts if the file already exists, unless kFileOpsWindowsDisablePrompt is passed in the iAppFlags parameter. (ST/JS/2976)

This makes the default behavior compatible with macOS (the file already exists prompt cannot be disabled on macOS).

FileOps.$readfile()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

FileOps.$readfile(*cFilePath,&vVariable*[,*iEnc*=kUniTypeAuto])

Description

Reads the file *cFilePath* into the variable *vVariable*. Returns an integer error code (zero for success).

cFilePath (Character 100000000) The pathname of the file.

vVariable (Variant) The variable into which the file contents will be read. Binary or Character. It it is Character, the file contents are converted to Character using the encoding specified by the iEnc parameter.

iEnc (Integer 32 bit, default is kUniTypeAuto) The encoding used to convert the file contents to Character. A kUniType… constant, not kUniTypeBinary or kUniTypeCharacter. Only used when *vVariable* is a Character variable.

FileOps.$writefile()

Function group	Execute on client	Platform(s)
FileOps	NO	All

Syntax

FileOps.$writefile(*cFilePath,vVariable*[,*iEnc*=kUniTypeUTF8,bBOM=kTrue,bReplace=kTrue])

Description

Writes the file *cFilePath* with the contents of variable *vVariable*. Returns an integer error code (zero for success).

cFilePath (Character 100000000) The pathname of the file.

vVariable (Variant) The variable into which the file contents will be read. Binary or Character. If it is Character, the file contents are converted to Character using the encoding specified by the iEnc parameter.

iEnc (Integer 32 bit, default is kUniTypeUTF8) The encoding used to convert the file contents to Character. A kUniType… constant, not kUniTypeBinary or kUniTypeCharacter. Only used when *vVariable* is a Character variable.

bBOM (Boolean, default is kTrue) If true, and character data is to be encoded as Unicode, $writefile adds a Unicode Byte Order Marker at the start of the file.

bReplace (Boolean, default is kTrue) Specifies what occurs if the file already exists before calling FileOps.$writefile(). If bReplace is true FileOps.$writefile() replaces the file; if bReplace if false, FileOps.$writefile() returns an error.

hexcolor()

Function group	Execute on client	Platform(s)
General	NO	All

Syntax

hexcolor(*string*)

Description

Returns a color value from the passed hex *string*. The hex *string* format is RRGGBB or RRGGBBAA for alpha support.

hsla()

Function group	Execute on client	Platform(s)
General	NO	All

Syntax

hsla(*hue,saturation,light*[,*alpha*])

Description

Returns a color value from the supplied color components: *hue* range is 0-360; *saturation* range is 0-100; *light* range is 0-100; optional *alpha* range is 0-255.

iconidwithbadge()

Function group	Execute on client	Platform(s)
General	NO	All

Syntax

iconidwithbadge(*cIcn,iCnt/cBadgeIcn*[,*kIconBadge*...,*iBadgeColor,iBadgeTextColor*])

Description

Returns a formatted string for $iconid. *cIcn* is the main icon (an SVG icon), *iCnt/cBadgeIcn* control the badge. The *kIconBadge*... constants control the display of the badge, and *iBadgeColor* and *iBadgeTextColor* set the colors of the badge and text.

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 parameters are:

Parameter	Description
cIcn	the ID of the primary SVG icon for the object / toolbar object
iCnt/cBadgeicn	the count to be displayed on the badge, or the ID of a smaller secondary icon
kIconBadge	kIconBadgeAlignTop, kIconBadgeAlignBottom, or the default is the position set by the OS, also kIconBadgeBackgroundHide, see below
ibadgecolor	the color of the badge, the default is kJSThemeColorSecondary
ibadgetextcolor	the color of the count, or secondary icon, the default is kJSThemeColorSecondaryText

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 *kIconBadge* 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 ))
```

isclient()

Function group	Execute on client	Platform(s)
General	YES	All

Syntax

isclient() *no parameters*

Description

Returns true if the code is currently running in JavaScript client-executed code.

iseven()

Function group	Execute on client	Platform(s)
Number	YES	All

Syntax

iseven(*integer*)

Description

Returns true if *integer* is an even number. If you pass a number rather than an integer, the function rounds it to the nearest integer before determining the result.

The function is available in client executed methods.

isodd()

Function group	Execute on client	Platform(s)
Number	YES	All

Syntax

isodd(*integer*)

Description

Returns true if *integer* is an odd number. If you pass a number rather than an integer, the function rounds it to the nearest integer before determining the result.

The function is available in client executed methods.

isoweekstart()

Function group	Execute on client	Platform(s)
Date and Time	NO	All

Syntax

isoweekstart(year,week)

Description

Returns the date of the first day of the specified ISO *week* in the specified *year*.

join()

Function group	Execute on client	Platform(s)
String	YES	All

Syntax

join(*list*[,*delimiter*=kTab,*column*=1,*stripWhite*=kFalse,*quoteChar*='"'])

Description

Returns a string of all column values in *list,* delimited by *delimiter* which defaults to kTab, but can be one or more characters. There are options to *strip* the leading and trailing whitespace (default is not to strip), and/or quote column values (escaping quotes as 2 quotes).

OIMAGE.$getdimensions()

Function group	Execute on client	Platform(s)
OIMAGE	NO	All

Syntax

OIMAGE.$getdimensions(*xImage, &iWidth, &iHeight* [,*&cErrorText*])

Description

Returns the dimensions in *iWidth* and *iHeight* of the image *xImage*. Returns Boolean true for success, or false and *cErrorText* if an error occurs.

xImage A binary variable containing the image.

iWidth and *iHeight* are 32 bit integer variables that receive the pixel Width and Height of the image.

cErrorText The error text returned from the function.

OIMAGE.$makeqrcode()

Function group	Execute on client	Platform(s)
OIMAGE	NO	All

Syntax

OIMAGE.$makeqrcode(*vData, &xQrCode* [,*iFmt*=kOIMAGEfmtPNG, *iSizePNG*=256, *iECL*=kOIMAGEerrHigh, *iBorder*=4, *&cErrorText*])

Description

Makes a QR Code for the specified binary or character vData. Returns Boolean true and xImage for success, or false and cErrorText if an error occurs.

vData: Binary or character data to be represented by the QR Code.The method converts character data to UTF-8 before encoding it. The amount of data that can be encoded is limited, and varies according to iECL. See the QR Code standard.

xQrCode: Binary variable that receives the generated QR Code

iFmt: A kOIMAGEfmt... constant that specifies the format of the generated QR Code. kOIMAGEfmtPNG or kOIMAGEfmtSVG

iSizePNG: The width and height in pixels of the QR Code, when generating kOIMAGEfmtPNG. Must be between 32 and 262144 inclusive

iECL: A kOIMAGEerr... constant that specifies the error correction level supported by the generated QR Code:

- **kOIMAGEerrLow**
  The QR Code can tolerate about 7% erroneous codewords
- **kOIMAGEerrMedium**
  The QR Code can tolerate about 15% erroneous codewords
- **kOIMAGEerrQuartile**
  The QR Code can tolerate about 25% erroneous codewords
- **kOIMAGEHigh**
  The QR Code can tolerate about 30% erroneous codewords

iBorder: The number of border modules to surround the QR Code

cErrorText: Receives the error text if an error occurs.

OIMAGE.$resize()

Function group	Execute on client	Platform(s)
OIMAGE	NO	All

Syntax

OIMAGE.$resize(*xImage, iWidth, iHeight, &xNewImage* [,*wParams*=#NULL, &cErrorText])

Description

Resizes the image *xImage* to the dimensions supplied in *iWidth* by *iHeight*. Returns Boolean true and *xNewImage* for success, or false and *cErrorText* if an error occurs.

xImage and *xNewImage* are both binary variables. xImage must be a JPEG or PNG image. xNewImage has the same type as xImage. You can use the same binary variable for both parameters if you want to replace the original image.

iWidth and *iHeight* can both be greater than zero and less than or equal to 32000, meaning resize to exactly that size. Alternatively, only one of the new dimensions can be zero, meaning calculate the dimension with value zero using the aspect ratio of the input image.

wParams is an optional row variable of parameters. The following columns can be specified in wParams:

- **sampler**
  The sampling method to be used when resizing. Either kOIMAGEsamplerBilinear or kOIMAGEsamplerNearestNeighbour. Defaults to kOIMAGEsamplerBilinear if this column is not present, or if wParams is omitted.
- **gray**
  A Boolean that indicates if the new image is to be a grayscale image. Defaults to kFalse if this column is not present, or if wParams is omitted.
- **quality**
  If the input image type is JPEG this column contains the JPEG image quality (1 to 100) of the new image. Defaults to 80 if this column is not present, or if wParams is omitted.

cErrorText The error text returned from the function.

OIMAGE.$transform()

Function group	Execute on client	Platform(s)
OIMAGE	NO	All

Syntax

OIMAGE.$transform(*xImage*,*&xNewImage*[,*wParams*=#NULL,*&cErrorText*])

Description

Perform a transformation on the image *xImage*. Returns Boolean true and *xNewImage* for success, or false and *cErrorText* if an error occurs.

xImage and *xNewImage* are both binary variables. xImage must be a JPEG or PNG image. xNewImage has the same type as xImage. You can use the same binary variable for both parameters if you want to replace the original image.

wParams is an optional row variable of parameters. The following columns can be specified in wParams:

- **transform**
  The transformation to be performed, a kOIMAGEtransform… constant, see below. Defaults to kOIMAGEtransformRotate90 if this column is not present, or if wParams is omitted.
- **quality**
  If the input image type is JPEG this column contains the JPEG image quality (1 to 100) of the new image. Defaults to 80 if this column is not present, or if wParams is omitted.

Valid transformations are:

Constant	Description
kOIMAGEtransformRotate90	Rotates the image 90 degrees clockwise; the default
kOIMAGEtransformRotate180	Rotates the image 180 degrees
kOIMAGEtransformRotate270	Rotates the image 270 degrees clockwise
kOIMAGEtransformFlipLeftRight	Flips the image on the left to right axis
kOIMAGEtransformFlipTopBottom	Flips the image on the top to bottom axis
kOIMAGEtransformTranspose	Transposes the image, i.e. flips the image along a 45 degree axis, from top-left corner to bottom-right corner

cErrorText The error text returned from the function.

ONOTIFY.$removebadge()

Function group	Execute on client	Platform(s)
ONOTIFY	NO	All

Syntax

ONOTIFY.$removebadge(*[,&cErrorText*])

Description

Removes the badge from the application icon. Returns Boolean true for success, or false (and sets *cErrorText*) if an error occurs.

ONOTIFY.$removelocal()

Function group	Execute on client	Platform(s)
ONOTIFY	NO	All

Syntax

ONOTIFY.$removelocal(*[vIDs,&cErrorText*])

Description

Removes local notifications with id(s) specified by *vIDs* which is a single character id, or a single column list of ids (remove all local notifications if *vIDs* is empty or omitted). Returns Boolean true for success, or false (and *cErrorText*) if an error occurs.

ONOTIFY.$sendlocal()

Function group	Execute on client	Platform(s)
ONOTIFY	NO	All

Syntax

ONOTIFY.$sendlocal(*cTitle, cMessage, vImage, iAction, wUserInfo,* [*iDelay*=0, *&cErrorText*])

Description

Sends a local operating system notification. Returns character notification id for success or returns #NULL (and cErrorText) if an error occurs.

If the call to $sendlocal() succeeds, it returns a character string. This is a string that uniquely identifies the notification. You can use this string to remove the notification from the system Notification Center using $removelocal, if for example the notification is no longer relevant.

The parameters are as follows:

Parameter	Description
cTitle	The title of the notification. Some text, displayed in bold font above the main notification text. The operating system will truncate this if it is too long. Windows allows this to occupy two lines, if you separate the lines using either kCr, lLf or kCr kLf. macOS only allows a single line
cMessage	The text of the notification. This is the main notification message, displayed in a plain font. The operating system will truncate this if it occupies more than 4 lines, either due to word wrapping, or the presence of newline characters (kCr, lLf or kCr kLf)
vImage	Image(s) to be displayed with the notification. See the 'Specifying Images' section
iAction	A value that specifies up to 2 optional actions that are to be included in the notification; on Windows, this is via one or two buttons; on macOS, this is either via a button for a single action, or via an options popup for two actions. A 'Specifying Actions' section
wUserInfo	A row containing user information that is passed to the $localnotify() method when the user clicks on the notification or a notification action. It must be possible to convert $userinfo to JSON. See section 'Handling Notification Clicks'

Parameter	Description
iDelay	The delay in seconds between the call to $sendlocal() and the notification being delivered (optional). Omnis can quit before the notification is delivered, as the operating system takes care of deferred delivery
cErrorText	A character variable that receives text describing an error if $sendlocal() fails

ONOTIFY.$setbadgecount()

Function group	Execute on client	Platform(s)
ONOTIFY	NO	All

Syntax

ONOTIFY.$setbadgecount(*iCount*[,*&cErrorText*,*iBadgeColor*,*iBadgeTextColor*])

Description

Sets the badge on the application icon to have the value *iCount*. Returns Boolean true for success, or false (and *cErrorText*) if an error occurs.

The parameters are as follows:

Parameter	Description
iCount	The count to display as the badge. Must be greater than zero. When running on Windows, a value greater than 99 is displayed as 99+.
cErrorText	A character variable that receives text describing an error, if $setbadgecount() fails
iBadgeColor	Windows only. The background color of the count badge. Defaults to styledbadgebackgroundcolor in the “system” section of appearance.json.
iBadgeTextColor	Windows only. The text color of the count badge. Defaults to styledbadgetextcolor in the “system” section of appearance.json.

ONOTIFY.$setbadgeicon()

Function group	Execute on client	Platform(s)
ONOTIFY	NO	All

Syntax

ONOTIFY.$setbadgeicon(*vIconId*[,*&cErrorText*,*iBadgeColor*])

Description

Note this is available on Windows only. Sets the badge on the application icon to be the specified icon *vIconId*. Returns Boolean true for success, or false (and sets *cErrorText*) if an error occurs.

The parameters are as follows:

Parameter	Description
vIconId	The icon id of the icon to display as the badge. The size

	component is ignored, as badges are always 16x16.
cErrorText	A character variable that receives text describing an error, if $setbadgeicon() fails
iBadgeColor	The color to be applied to the themed SVG; only applies if the icon is a themed SVG. Default is kColorHilight.

ord()

Function group	Execute on client	Platform(s)
Unicode	NO	All

Syntax

ord(*integer*[,*locale*,*thousands*=kFalse])

Description

Returns a string comprising positive *integer* with ordinal suffix for supplied *locale*, or current language if locale parameter is omitted or empty. Optionally includes ICU *thousands* separators, the default is not to include them.

OW3.$computername()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.$computername(*bFormat*=kFalse)

Description

Returns the name of the current computer or an empty string if the name could not be obtained. If *bFormat* is true, $computername attempts to format the string to make it suitable for an end-user, by removing a '.local' suffix, replacing all - characters with space, and capitalizing each word.

OW3.$parserfc3339()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

$parserfc3339(*cRfc3339*[,*bUTC*=kTrue,*&iOffset*,*&cErrorText*])

Description

Parses a date and time value conforming to RFC3339 and returns an Omnis date-time value and optionally the time zone offset in minutes. Returns #NULL if the string cannot be parsed.

The parameters are:

Parameter	Description
cRfc3339	a date and time string conforming to RFC3339
bUTC	If true, the returned date-time value is in UTC rather than the local time zone of the RFC3339 date-time value
iOffset	If the RFC3339 date and time string is parsed successfully this receives the time zone offset in minutes
cErrorText	If supplied, receives text describing the error that caused $parserfc3339 to return #NULL

OW3.$totpgenerate()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.$totpgenerate(*xSharedSecretKey,iTimeStep,iDigits,&iTOTP*[,*&cErrorText,iHashType*=kOW3hashSHA1])

Description

Generates a Time-based One-Time Password in iTOTP using the TOTP algorithm. Returns true if successful.

xSharedSecretKey (Binary) The shared secret key, length must be between 16 and 256 bytes inclusive.

iTimeStep (Integer 32 bit) The time step in seconds, must be between 1 and 3600 inclusive.

iDigits (Integer 32 bit) The number of digits in the TOTP, must be between 6 and 8 inclusive.

iTOTP (Integer 32 bit) Receives the generated TOTP.

cErrorText (Character 100000000) If supplied, receives text describing the error that caused $totpgenerate or $totpvalidate to return false.

iHashType (Integer 32 bit, default is kOW3hashSHA1) The type of hash to use, a kOW3hash… constant.

OW3.$totpvalidate()

Function group	Execute on client	Platform(s)
OW3	NO	All

Syntax

OW3.$totpvalidate(*xSharedSecretKey,iTimeStep,iDigits,iTOTP*[,*&cErrorText,iHashType*=kOW3hashSHA1,*iStepsBefore*=2,*iStepsAfter*=1])

Description

Validates the Time-based One-Time Password in iTOTP using the TOTP algorithm. Returns true if iTOTP is a valid TOTP.

xSharedSecretKey (Binary) The shared secret key, length must be between 16 and 256 bytes inclusive.

iTimeStep (Integer 32 bit) The time step in seconds, must be between 1 and 3600 inclusive.

iDigits (Integer 32 bit) The number of digits in the TOTP, must be between 6 and 8 inclusive.

iTOTP (Integer 32 bit) The TOTP that is to be validated.

cErrorText (Character 100000000) If supplied, receives text describing the error that caused $totpgenerate or $totpvalidate to return false.

iHashType (Integer 32 bit, default is kOW3hashSHA1) The type of hash to use, a kOW3hash… constant.

iStepsBefore (Integer 32 bit, default is 2) The number of time steps before the step for the current time that can be checked when validating the supplied TOTP, must be between 1 and 20 inclusive.

iStepsAfter (Integer 32 bit, default is 1) The number of time steps after the step for the current time that can be checked when validating the supplied TOTP, must be between 1 and 20 inclusive.

rgba()

The rgba() function can now be executed on the client, which allows you to set the color and alpha value of objects in client executed methods.

row()

The row() function now uses the variable name to name the columns if you pass in a variable. (ST/FU/806)

For example, if you pass in row(var1, var2), the row's column names would become var1 and var2.

startswith()

Function group	Execute on client	Platform(s)
String	YES	All

Syntax

startswith(*string*,*start*[,*ignorecase*=kfalse])

Description

Returns true if the *string* starts with the non-empty string *start*. If *ignorecase* is kTrue, the function uses case-insensitive comparison.

See also contains() and endswith().

Example

```
Calculate lString as 'Fast prototyping'

If startswith(lString,'Fast')
  # returned True
End If

Calculate lBoolean as startswith(lString,'FAST',kTrue)
# lBoolean = True
```

sys(251) and sys(252)

On macOS, sys(251) and sys(252) now return the overall width and height of all screens; in previous versions, these returned the main window area available on Windows. (ST/FU/737)

sys(254) and sys(255)

The functions sys(254) and sys(255) have been added to return the omnispdf folder and the omnispdf temp folder, respectively. (ST/FU/808)

sys(254) returns the pathname of the folder where Omnis writes scripts and other files used to generate PDF documents.

sys(255) returns the pathname of the folder where Omnis writes temporary PDF files.

sys(256) and sys(257)

The function sys(256) has been added to return the maximum server license count, and sys(257) to return the current number of server licenses in use. (ST/JS/3214)

sys(290)

The function sys(290) has been added to provide the method count excluding any cleared methods. (ST/FU/797)

sys(290) returns the number of methods on the method stack excluding any that are to be cleared. This does not work for client requests running in a thread of the Multi-threaded Server.

tracelog()

The tracelog() function now returns the logged string, or #NULL if the string could not be logged. (ST/DB/1149)

Commands

OK Message

If no icon is specified in the *OK message* command, the default action on macOS is to show the application icon (applies to Big Sur or later). (ST/HE/1748)

Set Timer Method

The *Set timer method* command now causes its timer method to run in the context of the task that was current when *Set timer method* was called. Note that the timer method will continue to run after the task closes. (ST/PF/1291)

Create Library

The *Create library* command now requires a full pathname for the new library file. (ST/PC/576)

Send to trace log

The "Start diagnostic logging" and "Stop diagnostic logging" options have been added to the *Send to trace log* command to allow you to start and stop the logging of diagnostic messages. (ST/DB/1390)

When either of these options is used with an empty log message parameter, no message is logged. The new options are as follows:

Start / Stop diagnostic logging
If specified, the command switches *on/off* the Log Diagnostic Messages trace log

option, before logging the message if the other command options allow. Also, if specified with an empty message to log, the command does not log an empty line.

The full syntax of the command is now:

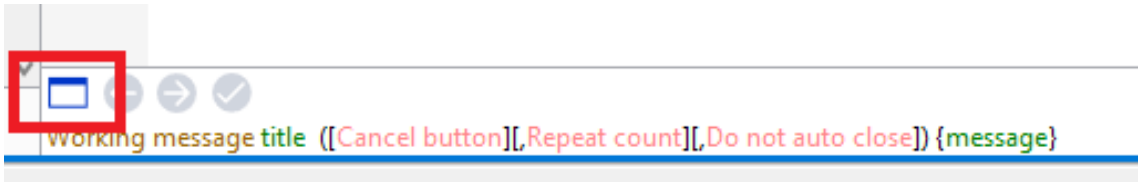
```
Send to trace log  ([Diagnostic message][,Always log][,Start diagnostic
   logging][,Stop diagnostic logging]) text
```

Working Message

The syntax for specifying the sequence of icons displayed in a working message has changed. In previous versions, you had to specify the *start* and *end* icon ID to use a consecutive series of icons in an iconset or icon datafile. Now you can use SVG icons, so the Working Message command can accept icon name IDs and now *all the icons used in the sequence* must be specified. For example, the new icons for the Generic (default) working message are specified as follows:

```
Working message
   Working/48:kIDEColorIcon:working01,working02,working03,working04,working05;5
   0;0;60
```

When configuring the Working Message command, you can click on the Helper button at the bottom of the Code Editor to open a helper window where you can set the parameters:



In addition, you can now specify the icon color to be used when using themed SVG icons. To specify a color, use either the name of an Omnis color constant, e.g. kRed, or a hex color, e.g. #0000FF which is blue. Set the color to kColorDefault to use the default theme color.

The previous syntax for configuring the icons is still supported for existing code, but if you edit a working message command with the old syntax, using the working message configuration Helper, the result is saved back to the code using the new syntax.

Deploying your Web & Mobile Apps

Headless Server Admin Tool

The Admin Tool for the Headless Omnis Server (osadmin) can now be used to upload new libraries and htm files to the /startup folder of the Omnis Server or to the webroot of the web server. (ST/AD/243)

On a Linux server, osadmin can now be used to 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.

The **Settings** section of osadmin can now take two extra options: 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.

Headless Server Serialization

Serialization for the Headless Omnis Server is now allowed using the OMNIS_SERIAL environment variable. (ST/SR/030)

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.

Version and Build Number

The **Version** and **Build** number of Omnis Studio has been added to the HTML generated by the **Test Form** option. (ST/JS/3123 & ST/JS/3125)

The "%%version%%" placeholder has been added to the template file (jsctempl.htm) which 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 has also been added to the template file which will be replaced with the Omnis Studio build number, e.g. 110034477 in the above example.

Omnis LSP Debugging

Debugging has been added to the **Load Sharing Process** (LSP) on the Omnis App Server. (ST/PF/1350)

You can enable debugging in the Omnis LSP using the new 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.

Web Server Plug-in ini

The web server plug-in ini (omnissrv.ini) file now works with the OMNISAPI plug-in to allow you to configure the plug-in. (ST/WT/1879)

Note that the ISAPI, CGI and Apache module, all look for omnissrv.ini in the same directory as the ISAPI/Apache DLL, or CGI exe.

oXML

Object References

There is a new property $useobjectrefs in the oXML external to force Omnis to return *Object References,* rather than objects. (ST/VR/324)

If true, the $useobjectrefs property ensures oXML returns object references rather than objects, that is, object return values are object references and object parameters must be object references.

This property is automatically set in new returned objects to the value of $useobjectrefs in the object returning the new object.

JavaScript Component SDK

JavaScript API Reference

Theme Methods

Static versions of the theme instance methods **getColorString()** and **getTextColorString()** have been added. The static versions are the same as the instance methods except you don't need a theme instance to call them.

The existing instance methods are retained for backwards compatibility, but they are deprecated in 11.0 and you should use the new static versions.

External Component SDK

GDI Reference

The **GDIcreatePixmapFromJPEG** function has been added. (ST/AD/252)

GDIcreatePixmapFromJPEG

GDIAPI HPIXMAP OMNISAPI GDIcreatePixmapFromJPEG(fldval& pPath, qdim& pWidth, qdim& pHeight)

Creates an HPIXMAP from JPEG image.

pPath - The pathname of the file containing the JPEG image

pWidth - If successful, receives the pixel width of the JPEG

pHeight - If successful, receives the pixel height of the JPEG

return - either the HPIXMAP for success, or NULL if an error occurs e.g. the file does not contain JPEG data

GDIAPI HPIXMAP OMNISAPI GDIcreatePixmapFromJPEG(qbyte pJpeg, qlong pJpegLen, qdim& pWidth, qdim& pHeight)*

Creates an HPIXMAP from JPEG image.

pJpeg - The address of the in-memory JPEG image

pJpegLen - The length in bytes of the in-memory JPEG image

pWidth - If successful, receives the pixel width of the JPEG

pHeight - If successful, receives the pixel height of the JPEG

return - either the HPIXMAP for success, or NULL if an error occurs e.g. the supplied data is not JPEG data.

PRI Reference

PRIdestParmStruct

The **PRIdestParmStruct** structure has been modified. Therefore, any xcomps that use PRIdestParmStruct must be rebuilt with the Studio 11 SDK. The following members of PRIdestParmStruct have been made private: mEdFile, mTextFile and mRepFile. As such, accessor methods have been added to the PRIdestParmStruct class:

void getRepFile(EXTfldval& pRetVal) const;
void setRepFile(EXTfldval& pNewVal);

void getTextFile(EXTfldval& pRetVal) const;
void setTextFile(EXTfldval& pNewVal);

void getEdFile(EXTfldval& pRetVal) const;
void setEdFile(EXTfldval& pNewVal);

These can be used by xcomps to retrieve and set the respective private members.

Deployment Tool

This section refers to the Deployment Tool for deploying your Windows or macOS desktop applications, not web and mobile apps.

A number of methods have been exposed in the Deployment Tool API to allow you to manage builds in your own code, rather than via the Deployment Tool UI. (ST/AD/228 and ST/AD/235)

Deployment Tool API

Using the new method $root.$modes.$getapiobject("customtool") Returns iObRef a number of API calls for the Deployment Tool have been exposed.

**$setcallbackinst**($cinst) takes a reference to an instance that implements $completed and $error. If the callback instance is set via this method, the deployment tool API will call either $completed or $error instead of returning the outcome to the caller. Note $error also receives a character variable as a parameter containing the error message and $completed can receive a 36-character long string on macOS if the built bundle is submitted for notarization.

$run(cConfigFilePath,cError[,cUUID]) requires the path to the deployment configuration file, a character variable to return errors to, or if on macOS, the UUID when the build is submitted for notarization. If successful, the method returns kTrue, otherwise kFalse is returned. The configuration file can be built using the GUI version of the Deployment tool.

Managing Builds via the API

The Deployment Tool API supports builds with in-memory data structures rather than file-based only. You can now get the data structures, load and save to a config.json file programmatically.

$getBuildDataStructure() returns a row containing the main data structure for a cross-platform build.

$getEntitlementsDataStructure returns a row containing two row: the standard and extended entitlements data structures (in that order).

$loadConfig(cPathToConfig.json, rBuildDataStructure, rEntitlements, cErrorText) takes in the path to a config.json containing data structures, a row that receives the build data structures, a row that receives and entitlements rows and a character variable that receives and error text. Returns true if successful, otherwise false.

$setBuildData(rBuildData) sets the build data structure in-memory to rBuildData row, or you can use $run without passing in the path to a build config.json in order to use the in-memory data values.

$setEntitlementsData(rEntitlements) sets the entitlements data structure in-memory to rEntitlements row (note the rEntitlements row must contain two rows where the first is standard entitlements and the second is extended entitlements. Works only on macOS.

$saveConfig(cPathToFile, bOverwrite, cErrorText) saves the build data and entitlements data currently stored in the API object to a .json file in cPathToFile. If cPathToFile already exists, bOverwrite (defaults to false) will be used to determine if the file should be overwritten. cErrorText receives any errors if unsuccessful; function returns true if successful, otherwise false.

The $run function can be run without passing the path to a build config.json file, e.g. Do api.$run("",cError, cUUID) Returns bOutcome as long as the build data has been set via $setBuildData, the build can start. You can use $run by passing the path a build config.json and the build/entitlements stored in the file will be used.

Creating config.json in the UI

The Deployment Tool allows you to create your own config.json file from inside the UI itself to provide settings for your application package. (ST/AD/237)

To create your own Configuration file you need to enable the Custom config.json option on the Additions tab in the Deployment tool. A customtool folder is created within the installed writable directory of Omnis containing a copy of the config.json file from the current instance of Omnis. From there you can edit the configuration file for your application to build. Alternatively, you can select an existing config.json to use for your application.

Removing Items from Builds

You can now remove files or folders during a build by specifying them on the Size Optimization tab. (ST/AD/211)

When adding files or folders to be removed, you only need to specify the relative path to the file or folder inside the bundle (macOS) or readonly/readwrite directory (Windows).

The tab will give some information regarding the estimated size before the build and the estimated size saved.

oProcess

oProcess is a new Worker Object (external component) providing a simple interface to launch and manage other processes, executables and applications, thereby providing you with greater interoperability from within Omnis Studio.

You can interact with oProcess using the standard worker methods, e.g. $init(), $run(), etc, which are described below, plus the external component has the common worker properties.

Properties

The OProcess object has the following properties:

Property	Description
$callbackinst	Sets the instance that will receive a worker's callbacks.
$cancancel	If kFalse, you can only cancel the worker forcefully. Defaults to kTrue.
$elapsed	Seconds elapsed since the worker's process was launched. Stops counting when the process returns an exit code.
$pid	The worker's process identification.
$exitcode	The worker's process exit code.
$timeout	Seconds the worker's process is allowed to run before getting cancelled. Defaults to 0 (no timeout).
$eol	End-of-line character which when encountered, a callback to the appropriate stream the worker's process wrote to is executed. Defaults to kLf for Linux and macOS and kCr,kLf for Windows. Setting $eol to an empty string i.e. $eol.$assign("") will cause an immediate callback to the stream the worker's process has written to.
$state	Returns the worker's current state.

$errortext	Returns the error text associated with the last action.
$threadcount	Returns the number of active background threads for all worker instances.
$errorcode	Error code associated with the last action.

Methods

$init()

$init(cProcess [,rArguments, cInitialDirectory, lEnvironment])

Initialises the worker to launch process in *cProcess* parameter. Use the *rArguments* row parameter to pass arguments to the process. *cInitialDirectory* can be used to launch the process with a different current directory. *lEnvironment* is a two column list of environment variables and their values to be used during the process' runtime. For example, launching the following process:

proc.$init('/bin/echo$TEST',,,list(row("TEST","Hello world!")))

will result in callback to $stdout with "Hello world!" in the stdout column.

$run()

Runs the process on the worker's main thread, therefore blocking code execution until the process returns. Should be avoided, unless there are specific synchronous requirements.

$start()

Starts the process on the worker's background thread (non-blocking).

$cancel()

$cancel([bForce=kFalse])

Cancels the worker's process. Pass kTrue for bForce parameter to close the process forcefully (currently supported only on Linux and macOS, sends SIGTERM signal). If bForce is kFalse, a SIGINT signal is sent. Note: if $cancancel property is kFalse and bFore is kFalse, the call to $cancel will be ignored: use kTrue for bForce to override the $cancancel property.

$completed()

$completed(wResults)

Callback method when the worker has finished running. wResults is a row with a retcode column containing the return code of the process and runtime_seconds column containing the seconds the process was alive for.

$cancelled()

Callback method when the worker's process has been cancelled.

$started()

Callback method when the worker's process has started and can now write to stdin. You can use this callback to work with processes that expect input as soon as they start running, e.g. when they prompt for a password.

$isrunning()

Returns kTrue if the worker's process is running, meaning that is has a PID greater than 0.

$stdout()

$stdout(wResults)

Callback method when worker's process writes to the stdout stream. wResults is a row with a stdout column containing the text the worker's process has written.

$stderr()

$stderr(wResults)

Callback method when worker's process writes to the stderr stream. wResults is a row with a stderr column containing the text the worker's process has written.

$write()

$write(cCharacters)

Writes cCharacters to the stdin stream of the worker's process.

$readlines()

$readlines(iStream [,nLines=0])

Returns a list containing all the lines written to kOProcessStd... stream, starting from the beginning of the stream. Use optional parameter nLines to limit the number of lines returned. For example:

$readlines(kOProcessStdin, 3)

will return the first 3 lines of the stdin stream.

iStream for $readlines() and $readtail() can be one of the following constants:

kOProcessStdin	Identifier for the stdin stream.
kOProcessStdout	Identifier for the stdout stream.
kOProcessStderr	Identifier for the stderr stream.

$readtail()

$readtail(iStream [,nLines=0])

Returns a list containing all the lines written to kOProcessStd... stream, starting from the end of the stream. Use optional parameter nLines to limit the number of lines returned. For example:

$readlines(kOProcessStdout, 3)

will return the last 3 lines of the stdout stream.

Using oProcess

Using the $init method you can run multiple bash commands as follows:

```
proc.$init("/bin/bash",row("-c","echo hey && echo hey2"))
```

or without using the arguments parameter

```
proc.$init("/bin/bash -c 'echo hey && echo hey2'")
```

On macOS and Linux, you can run processes as the root user as follows:

```
proc.$init("/usr/bin/sudo",row("-S","/usr/bin/whoami"))
```

and when the $started callback is received, call proc.$write(con("password",kLf)) to respond with the password. In this case, you will receive a call to $stdout with the stdout column containing "root", indicating that process is running with higher privileges.

On Windows, you cannot elevate the currently running process since the underlying APIs that make use of RunAs cannot redirect the stdout and stderr streams, suggesting that you cannot directly capture the output streams of an elevated process from a non-elevated process. Although the best way to ensure the elevated privileges are transferred to the process launched is to run Omnis with elevated privileges, you could do:

```
proc.$init('powershell.exe start powershell -Verb runAs -ArgumentList \"net
   session\" -WindowStyle hidden -Wait')
```

to execute something as admin when Omnis is not running as admin, but you will not get the $stdout or stderr callbacks and you will not be able to use $write to the elevated process, making it a run-and-forget process.

Appendix

Omnis Configuration Items

The following is a complete list of configuration groups and items in the Omnis Configuration file (config.json) which you can edit using the **Edit Configuration** option in the Studio Browser (click on the Options button in the bottom-left of the Studio Broswer).

Not all of these items appear in the default config.json provided with Omnis Studio, but you can add any item in the Configuration Editor; you must use the exact spelling and case of the item when adding it.

codeAssistant

The items in the **codeAssistant** group configure the behavior of the Code Assistant.

createVariableScopePrefixes

Use this configuration item to configure the scope suggested for new variables created by the Create Variable dialog, based on the prefix of the variable name.

You can add a new variable simply by typing its name in a code line and declaring the variable in the Create Variable dialog. When you type the name of the new variable in your code, initially it will not be recognized and is marked as an error. In this case, clicking on the Fix button at the base of the Code Editor window, or the fix error keyboard shortcut, will open the Create Variable dialog, allowing you to declare the new variable, including its scope, data type, subtype, initial value and description.

Note that the unrecognized variable dialog can also open when assigning a new or unknown variable name to a property in the Property Manager. In this case, for properties such as $dataname, the initial type of the variable creation dialog is set to the most likely data type for the control, e.g. List data type for a list form control. The dialog restricts the scope of the new variable to what makes sense based on class type, and so on.

When you type the name of a new variable in your code, you can specify the initial scope for the variable using a predefined prefix; the Create Variable dialog will select the scope automatically. For example, you could begin the variable name with "i" to create an instance variable, or "p" to create a parameter. You configure these prefixes using the **createVariableScopePrefixes** item. This is an array, where each entry has the syntax

```
prefix:scope
```

where **prefix** is the case-insensitive prefix that results in the specified scope, and **scope** can be the string Instance, Class, Parameter, Local or Task.

The Create Variable dialog processes these entries in array order, and as soon as it finds a scope that is allowed for the method being edited (e.g. instance variables are only allowed for class types that have instances), where the first part of the entry value case insensitively matches the start of the variable name, it uses the configured scope (the second part of the entry value after the colon) to set the initial scope suggested by the dialog. If no prefix match occurs, the scope suggested is local.

createVariableTypeSuffixes

Use this item to configure the type suggested for new variables created by the Create Variable dialog, based on the suffix of the variable name. See the description of **createVariableScopePrefixes** for details of the Create Variable dialog.

createVariableTypeSuffixes is an array, where each entry has the syntax

```
suffix:type
```

where **suffix** is the case insensitive suffix that results in the specified type, and **type** is a type constant name such as kList, kItemref, kDate, kObject or kBinary.

Omnis strips any consecutive digits from the end of the desired variable name, and then compares (case independently) the end of the resulting name string against the suffixes in the array. If there is a match, and if the variable type is suitable (e.g. it is not a non-client executed type when creating a variable for a client-executed method), then the initial type is set using the type constant after the colon.

currentCommandFilter

The name of the current command filter selected by the Code Editor. This is automatically set by the Code Editor when executing Save Window Setup for the Code Editor window.

listShowsNamesFirst

Boolean. Default true. If true, notation attributes appear last in code assistant lists. So for example, when showing the list for $objs, the object names occur in the list before the group methods for $objs.

maxParameterHelpWidth

Integer. Default two thirds of the screen width. Specifies the maximum pixel width of the parameter help popup.

oldSortOrder

Boolean. Default false. If false, the code assistant list is sorted with names and variables first, followed by functions, then notation attributes, then constants and then events. Each set of entries for a particular entry type is sorted by entry name.

If true, the code assistant list is sorted in the same way as it was in Studio 10.2. In this case, the listShowsNamesFirst config item also applies.

oldTabReturnBehavior

Boolean. Default false. Applies when the code assistant popup is open. If false pressing tab extends the current text with which the code assistant is working to the longest matching prefix, whereas return replaces the current text with the first item in the code assistant list. If true, tab behaves like return.

openParameterHelpWithCodeAssistantPopup

Boolean. Default true. If true, the code assistant and parameter help window both open on the same side (above or below the text being entered). If false, they open on opposite sides.

parameterHelpEnabled

Boolean. Default true. Enables or disables the code assistant parameter help popup.

parameterHelpSpace

Integer. Default 40 pixels. The space (in pixels) for parameter help on the same side of the text as the code assistant popup; applies when **openParameterHelpWithCodeAssistantPopup** is true.

tabAlsoLeavesFieldAfterClosingAssistant

Boolean. Default false. Ignored unless oldTabReturnBehavior is false. If true, and the current field is not the code entry field in the code editor, it affects the tabbing behavior in the code assistant: a tab will close the code assistant and the cursor will move to the next field in the tabbing order.

useOmnisHelpPagesForFunctionHelp

Boolean. Default true. If true, the code assistant uses Omnis help pages to display the content of the help panel for functions. If false, the code assistant uses a short text description.

width

Integer. Default 768. The pixel width of the code assistant window. Must be between 512 and 1536 inclusive.

complexgrid

The items in the **complexgrid** group configure the behavior of the fat client complex grid.

mingridpos

Integer. Default zero. Specifies how close grid lines in a fat client complex grid can be to one another. Zero allows them to overlay each other, giving the appearance of hidden columns. A positive value allows them to be further apart. A negative value is treated as zero.

shiftRequiredToResizeAllRows

Boolean. Default true. Controls the way complex grid rows are resized when using the mouse.

If true, dragging a row divider without pressing the shift key resizes the single row above the row divider. To make all rows have the new row height, press the shift key while dragging a row divider.

If false, the behavior is reversed. Press the shift key while dragging a row divider in order to resize the single row above the row divider. Dragging a row divider without pressing the shift key gives all rows the new row height.

debugger

The items in the **debugger** group configure the behavior of the local and remote debugger.

autoVariablesContextAttributes

An array of character strings that specify the context attributes to display on the auto tab of the debugger variable panel. Default empty, which means the panel includes $cinst, $cobj and $ctask. If not empty, each array member must be a context attribute name e.g. "$cinst".

defaults

The items in the **defaults** group configure various default behaviors of the Omnis environment.

allExternalComponentErrorsAreFatal

Boolean. Default true. If true, errors reported in external components by setting #ERRCODE and #ERRTEXT, always generate a runtime error.

disableAllLibraryConversionPrompts

Boolean. Default false. If true, all library conversion prompts are disabled, and libraries convert without any prompts.

dropDestinationFrameAlpha

Integer (0-255). Default 192. The alpha value used when drawing the drag and drop destination rectangular frame.

dropDestinationFrameColor

Integer. Default colorhighlight system color defined in appearance.json. The color used when drawing the drag and drop destination rectangular frame.

dropDestinationFrameWidth

Integer. Default 4. The width in pixels of the drag and drop destination rectangular frame.

enableCrashReporting

Boolean. Default true. If true, crash reporting is enabled, meaning that if Omnis crashes, it writes a minidump file to the logs/crashes/reports folder.

entryFieldsIncludeQuotesWhenSelectingWords

Boolean. Default false. If true, fat client entry fields treat single and double quote characters as part of words.

extraClassNameValidations

Boolean. Default true. If true, Omnis performs extra validations before assigning a name to a class. These validations are recommended, as using the characters they exclude in class names can cause confusion and potentially errors. This property is present to allow code from previous versions to continue working if the additional validations cause the code to fail.

The characters excluded by these validations are any character with a value less than space, a character in the string ".,;:!?)]}([{+-*/|&><=", or a single or double quote character. In addition, the validations do not allow the name to start with the character $.

floatWindowSubclass

Boolean. Default true. If true, and the subclass overrides $width or $height of its superclass, apply appropriate floating to superclass controls based on their $edgefloat value.

initiallayoutbreakpoints

A comma-separated layout breakpoint character string that specifies the value assigned to $clib.$prefs.$initiallayoutbreakpoints when creating a new library. Default 320,768 if empty.

language

Default empty. If not empty, a character string that specifies the name of a language in the localization data file, used to override the current language set in the localization data file.

maxCachedIconSetBitmaps

Integer. Default 1000. Configures the cache of bitmaps generated from PNG and SVG images held in icon sets. The maximum number of bitmaps that can be cached. 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 frees the least recently used bitmap.

maxChainDepth

Integer. Default 20. Configures the maximum number of field or item references that Omnis will chain through in order to reach the actual referenced variable. Normally you would leave this item set to 20. Change it if you have a heavily recursive method that uses field reference parameters.

Note the minimum value is 20 so that setting this to any value less than 20 results in Omnis using the value 20.

nationalFieldCompareChars

Boolean. Default false. Controls how national fields are compared.

If true, national fields are compared a character at a time. If false, national fields are compared using the ICU collator. You are recommended to leave this value set to the default, unless advised otherwise by Omnis Support.

Note that if you are using the Omnis data file and you change this value, you will need to drop and rebuild all indexes.

omnisPreviewURLPrefix

The report preview URL prefix for $address; defaults to "omnis:" if empty.

reportErrorOpeningInitialFileAsLibrary

Boolean. Default true. If true, Omnis reports an error when trying to open the initial file (dropped on Omnis, or double clicked) as a library. If false, Omnis ignores an error when doing this.

reportQueueCommandFieldNotFoundErrors

Boolean. Default true. If true, queue commands (such as Queue set current field) generate a debugger (or runtime) error if the field cannot be found. If false, execution just continues if the field cannot be found; setting this item to false can be used to provide compatibility with earlier versions of Omnis.

responsiveLayoutPadding

Integer. Default 2. The default value of $layoutpadding for a new remote form.

showLibraryConversionWorkingMessage

Boolean. Default true. Specifies if a working message can be displayed to show the progress of conversion of a library from an earlier version of Omnis to the current version.

stricterNotationErrorChecks

Boolean. Default true. When true, certain unresolved name errors (for example, from notation of the form $cinst.name and $ctask.name) result in a debugger (or runtime) error if $clib.$prefs.$reportnotationerrors is kTrue.

sys192ExcludesIDEmethods

Boolean. Default true. Specifies whether to exclude IDE method calls from the list returned from sys(192). If true, it excludes an IDE method if the library containing the method is marked as always private.

sys192ListRowLimit

Integer. Default zero. If greater than zero, lists (and rows) with up to **sys192ListRowLimit** rows are included as a third column in the parameter data. Each parameter in the parameter list stored in each line of the sys(192/292) list has a third column, which for lists and rows contains the actual list (or row) data, if the list or row has less than or equal to **sys192ListRowLimit** lines. In all other cases (not a list or row, or line limit **sys192ListRowLimit** exceeded) column 3 is empty.

If less than or equal to zero, no third column is added.

tokenEntryPopupDelay

Integer. Default 500. The delay in milliseconds after you stop typing in the fat client token entry field, after which the token entry popup can appear.

tokenizeExternalFieldNames

Boolean. Default false. The initial value for $prefs.$tokenizeexternalfieldnames. If true, Omnis uses tokens rather than text when tokenizing external field names.

tokenizeExternalFileNames

Boolean. Default false. The initial value for $prefs.$tokenizeexternalfilenames. If true, Omnis uses tokens rather than text when tokenizing external file names.

traceLogUsesStyles

Boolean. Default true. Specifies whether the trace log renders text styles.

Notes: If false, trace log lines can still contain text styles; in this case, Omnis ignores the styles when drawing trace log lines. Omnis always strips text styles from trace log lines written to the text log file in the logs folder, irrespective of the value of traceLogUsesStyles. (In effect this replaces traceLogUsesSyntaxColors which has been removed.)

useOldRegularExpressionSyntax

Boolean. Default false. If false, Omnis uses PCRE2 compatible regular expressions. If true, Omnis uses the regular expression syntax used in Omnis Studio version 10.1 and earlier.

useScreenDestination

Boolean. Default false. If false, Omnis automatically maps the old Screen report destination to the Preview report destination. If true, the old Screen report destination is available.

useSystemSettingsForSpelling

Boolean. Default true. If true, entry fields that perform spelling checks and provide spelling suggestions, use the system settings to identify the current language or languages. If false, these entry fields use the national sort ordering locale for the current language in the Omnis localisation data file. If Omnis fails to initialise the system spelling API to use the required language it reports this failure to the trace log.

diacriticalpopup

The diacritical popup opens after you long press on certain keys in fat client entry fields, giving you the option of entering related characters with diacritical marks for either the current Omnis language, or optionally on macOS the language determined from the current keyboard layout.

Use items in this group to configure the behavior of the diacritical popup on macOS.

The items in the **diacriticalpopup** group are diacriticalPopupUsesMacOSKeyboardLayout and keyboard identifiers, for example:

"com.apple.keylayout.British": "en",

"com.apple.keylayout.German": "de",

"com.apple.keylayout.French": "fr",

"com.apple.keylayout.Spanish": "es",

"com.apple.keylayout.Italian-Pro": "it",

"com.apple.keylayout.Italian": "it"

Each keyboard identifier is the string id of a macOS keyboard layout. The value of each keyboard identifier item is the language identifier - this is the language used for the diacritical popup when the current keyboard layout matches the item name.

diacriticalPopupUsesMacOSKeyboardLayout

Boolean. Default true. Only applies to macOS. If true, Omnis tries to determine the language for the diacritical popup by using the current keyboard layout and the remaining items in the diacritical popup group.

docview

The items in the **docview** group configure the behavior of the docview external component.

replaceTabsInRTFwithSpacesWhenAddingToReport

Integer. Default 2. This can be a value from zero to 32 inclusive, and it specifies what happens to tab characters in RTF when the RTF object is added to a report. Zero means leave the text unchanged. 1-32 means replace each tab character found in the text with 1-32 spaces.

exportimportjsonoptions

The **exportimportjsonoptions** group stores the values of the $prefs.$exportimportjsonoptions Omnis preference.

deleteexportoutputtreeifcancelled

Boolean. Default true. If true, $exportjson() deletes a partially exported output tree, if the export is cancelled by the user.

exportcodefoldingstate

Boolean. Default false. Controls whether the code-folding state in the methods in your library is exported by $exportjson().

exportoverwritesconflicts

Boolean. Default false. If false, $exportjson() does not replace the folder for a conflicting class, and before it exports the class, renames the folder for the class to be of the form name.conflict<n> where <n> is an optional integer added if the .conflict folder already exists from a previous export; this makes both the original JSON and the new JSON available in the library tree.

If true, $exportjson() replaces conflicts when overwriting an existing tree (conflicts are ignored when the method text file extension has changed, or class.json contains the error marker - in other words, in these cases, the class is always replaced).

hideexportworkingmessage

Boolean. Default false. If true, the working message is hidden for $exportjson() and $comparejson().

hideimportworkingmessage

Boolean. Default false. If true, the working message is hidden for $importjson().

importtreatsunknownpropertyaswarning

Boolean. Default true. Specifies whether unknown properties in imported JSON are treated as a warning or an error.

includenotationinide

Boolean. Default false. Specifies whether the notation methods $comparejson(), $exportjson() and $importjson() are present in the IDE property manager tables. This also controls whether these notation methods appear in the code assistant.

ide

The items in the **ide** group configure the behavior of the IDE.

allowNumericObjectNames

Boolean. Default false. When false, the Property Manager validates the value assigned to $name for menu, remote form, remote menu, report, schema, toolbar and window objects. The validation applied is that when the name starts with a digit, the remaining characters cannot all be a digit or a character in the string "+-.".

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.

animateIDEcontrols

Boolean. Default true. Specifies whether controls on IDE windows can be animated.

autoSave

Boolean. Default false. The state of the Auto Save option on the file menu. Omnis saves the state of this option selected via the file menu in this configuration item.

When Auto Save is enabled, Omnis periodically automatically saves the content of all class and code editors, except for system classes, provided that the class is not read-only, and the code editor is not in read-only mode.

autoSaveInterval

Integer. Default 1000. Applies when Auto Save is enabled. The interval between each Auto Save in milliseconds.

canUseCreateVariableOnVarNotFound

Boolean. Default true. Specifies if the Create Variable dialog can be used to define a missing variable.

catalogUsesSyntaxColors

Boolean. Default true. If true, the Catalog displays items in the right hand list using their syntax color (if any).

componentStorePopupDelay

Integer. Default -1. The delay in milliseconds between a click on an entry in the Component Store window and the popup for that entry appearing. -1 means Omnis calculates the delay to be just longer than the double click time, which means you can double click on an entry to add the corresponding default component to the design window without the popup appearing briefly.

currentJavaScriptTheme

The name of the current JavaScript theme. Stores the value of the property $prefs.$javascripttheme.

dockingAreaDesignDPI

The scaling DPI values used for the main docking areas when **dockingAreaDesignDPIMode** is not set to kDPIoff. A comma separated string of three DPI values. Default 96,72,75.

dockingAreaDesignDPIMode

A character string specifying the main docking area DPI mode. Default kDPIoff. Either kDPIoff, kDPIframeOnly or kDPIall.

dragObjectAlpha

Integer. Default 220. The alpha value (0-255) applied to the drag bitmap when dragging objects from a fat client window.

findAndReplaceLogUsesSyntaxColors

Boolean. Default true. Specifies whether the find and replace log uses syntax colors when displaying method lines.

findAndReplaceSelectsTopClass

Boolean. Default true. If true, when the find and replace dialog comes to the front, it selects the class of the top-most window that has a class. If false, or the dialog cannot find a class (because for example there is no window with a class), then the selection remains unchanged.

libConverterAddsInlineCommentToStaCommandParameter

Sta: commands in Omnis Studio 10.0 and later cannot have an inline comment. This item configures how to convert an inline comment on a Sta: command, when converting from versions prior to Omnis Studio 10.0.

If empty, Omnis does not add the inline comment to the Sta: command parameter. The inline comment is discarded, and then processed according to **libConverterAppendsDiscardedInlineComments** and **libConverterInsertsDiscardedInlineComments**.

If not empty, it must be a string containing a % placeholder e.g. "**-- %**" or "**/* % */**". When converting a Sta: command with an inline comment, Omnis replaces the % with

the inline comment, appends it to the Sta: command parameter, and attempts to tokenize the command. If the command tokenizes, conversion is complete; otherwise, the inline comment is discarded, and then processed according to **libConverterAppendsDiscardedInlineComments** and **libConverterInsertsDiscardedInlineComments**. If **libConverterAddsInlineCommentToStaCommandParameter** does not contain %, the inline comment is similarly discarded.

libConverterAppendsDiscardedInlineComments

Boolean. Default true. If true, an inline comment from a Text: or JavaScript: command, or an inline comment discarded from a Sta: command (see **libConverterAddsInlineCommentToStaCommandParameter**) is appended as a comment after the command.

libConverterInsertsDiscardedInlineComments

Boolean. Default false. If true, an inline comment from a Text: or JavaScript: command, or an inline comment discarded from a Sta: command (see **libConverterAddsInlineCommentToStaCommandParameter**) is inserted as a comment before the command.

This item is ignored if **libConverterAppendsDiscardedInlineComments** is true.

maxDisplayedDropListLines

The maximum number of displayed lines in all droplists and combo boxes in the Omnis IDE. Defaults to 30, and can be 5-50 inclusive.

maxRecentClassEntries

Integer. Default 9. Can be set to any value in the range 9 to 32 inclusive. Specifies the maximum number of recent class entries in the View menu on the main menu bar. Note that this also affects the class browser recent classes hyperlink, but since that only shows classes (or methods when the shift key is pressed), there are typically fewer recent class items on the recent classes hyperlink than on the main View menu.

neverUseSystemStyleTooltips

Boolean. Default false. If true, Omnis does not use system style tooltips (ignoring the setting of the systemstyle entry in the tooltip section of appearance.json).

positionAssistantKeyboardTimer

Integer. Default 750. Used when position assistance appears while using the keyboard. Specifies the maximum time that position assistance remains visible after you stop pressing an arrow key.

positionAssistantShowsPositionOrSize

Boolean. Default true. Applies to position assistance for the remote form and window editors. Specifies whether the position assistant displays the current position when objects are being moved or dropped from the Component Store, or the current size when objects are being sized. When more than one object is selected, the position or size corresponds to the union of the object rectangles.

restoreOpenClassEditorsAtStartup

Boolean. Default true. If true, after completing startup, the development version of Omnis tries to re-open class editors that were open when Omnis last shut down successfully. Note that the class editors to which this applies do not include the system table editors.

Note: The value of this configuration entry is ignored, and treated as false, if the restoreOpenLibsAtStartup entry is false.

restoreOpenLibsAtStartup

Boolean. Default true. If true, after completing startup, the development version of Omnis tries to re-open libraries that were open when Omnis last shut down successfully.

When the development version of Omnis shuts down successfully, it saves the list of libraries to re-open. The library list saved excludes all libraries in the startup and studio folders, and all private libraries; these libraries will typically re-open anyway. In addition, Omnis will only run the startup task of a library that it re-opens, if the startup task was open when Omnis last shut down successfully.

saveSearchDelay

Integer. Default 500. Each keystroke in an IDE search combo box performs a search. After starting a search, if **saveSearchDelay** milliseconds passes without a keystroke, the search combo box saves the current search string for future use via the combo box popup list.

saveWhenSettingGoPoint

Boolean. Default false. If true, a save is triggered when manually setting the Go point while debugging your code.

searchFindAndReplaceLogByType

Boolean. Default true. Controls how keyboard searches work in the find and replace log. Only applies when $prefs.$oldlistsearching is false. When true, and $prefs.$oldlistsearching is false, keyboard searches in the find and replace log search column 2 (the type column) rather than column 1.

syntaxColorProbableSQLComments

Boolean. Default true. If true, the code editor attempts to recognize and syntax color comments in Sta: commands, by looking for -- and /* */ comments.

tryDesignTaskWhenTestingWindow

Boolean. Default true. If true, when testing a window using Ctrl/Cmd+T, Omnis looks at the design task name, and if it is the same as the startup task name, switches to the startup task. If however, the design task name is different, Omnis switches to the first instance it can find of that task; if there is none, it switches to the startup task as before.

If false, when testing a window using Ctrl/Cmd+T or the open window hyperlink of the browser, or the browser context menu for a window class, Omnis switches to the startup task of the library containing the window.

updateVariableWindowsOnDebuggerStop

Boolean. Default true. If true, Omnis updates any open variable value windows with the current variable value, when stopping in the local debugger. If false, the update does not occur in a variable window until it becomes the top window.

windowToolbarDesignDPI

The scaling DPI values used for the window toolbar of IDE windows when **windowToolbarDesignDPIMode** is not set to kDPIoff. A comma separated string of three DPI values. Default 96,72,75.

windowToolbarDesignDPIMode

A character string specifying the DPI mode for the window toolbar of IDE windows. Default kDPIoff. Either kDPIoff, kDPIframeOnly or kDPIall.

java

The items in the **java** group configure Java support when Java support has been included as part of the Omnis installation. (Note: the Omnis Java support files are not installed by default, and need to be added only if required.)

jvmPath

The pathname of the JVM DLL used when Java support has been included as part of the Omnis installation.

resetClassCacheOnStartup

Boolean. Default false. If true, and Java support has been included as part of the Omnis installation, the Java objects external component discards the class cache when Omnis starts.

startjvm

Boolean. Default false. Only used by the headless Linux server. If true, and Java support has been included as part of the Omnis installation, start the JVM during headless server startup.

log

The items in the **log** group configure logging.

conversionLogDelimiter

Configures the delimiter used by the log generated in the logs/conversion folder, when converting a library from a version prior to Omnis Studio 10.0. Default \t. You can enter \t to mean tab, or any single character in this item.

datatolog

An array of keys that specifies the information that will be written to the log. Supported keys are:

restrequestheaders	RESTful request headers
restrequestcontent	RESTful request content
restresponseheaders	RESTful response headers
restresponsecontent	RESTful response content
tracelog	Trace log
seqnlog	Sequence log
soapfault	SOAP fault
soaprequest	SOAP request
soapresponse	SOAP response
soaprequesturi	SOAP request URI
cors	RESTful CORS processing errors
headlessdebug	Headless server event handling debug messages
headlesserror	Headless server event handling error messages
headlessmessage	Headless server general messages

encloseConversionLogTextInQuotes

Boolean. Default true. Specifies whether entries in the log generated in the logs/conversion folder, when converting a library from a version prior to Omnis Studio 10.0, are enclosed in double quotes.

logcomp

Either empty (meaning logging does not occur) or the name of the logging component to use for logging. Omnis currently only provides a single logging component named logToFile.

logDamSerialNumberErrors

Boolean. Default false. If true, Omnis writes an entry to the trace log if it fails to load a DAM because the serial number does not allow the DAM to be used.

logNowExternalComponentErrors

Boolean. Default true. If true, Omnis writes an entry to the trace log if it fails to load an external component because either the component was built for Studio Now and the current Omnis build is a periodic release, or the component was built for a periodic release and the current Omnis build is a Studio Now release.

logToFile

An object that configures logging when the **logcomp** item is set to logToFile. The object has members as follows:

folder	The name of the folder in which log files are written. The name must not be empty, and it must not end in a path separator. It can either be a full pathname or a path relative to the Omnis data folder. When using a full pathname, the folder must exist; when using a relative pathname, the logToFile component attempts to create the log folder (including any intermediate folders).
rollingcount	The number of log files to keep. logToFile starts a new log file every log period (see the **daily** item). **rollingcount** specifies the maximum number of log files that can be kept in the log folder. When the limit is reached, and logToFile needs to start a new log file, logToFile deletes the oldest log file. rollingcount must be between 2 and 1024 inclusive.
daily	logToFile starts a new log file every log period. A log period is either hour or day. Use this Boolean item to specify which. When set to hour (**daily** = false), a new log starts at startup, and when a log entry is to be generated after the hour has changed. When set to day (**daily** = true), a new log starts when the day has changed; in this case, when you start Omnis, and there is already a log file for the current day, logToFile appends to it.

overrideWebServicesLog

Boolean. Default false. SOAP Web Service support in Omnis has a separate log file.

When **overrideWebServicesLog** is false, SOAP log messages are written to both the **logcomp** log and the separate log file.

When **overrideWebServicesLog** is true, SOAP log messages are only written to the **logcomp** log.

windowssystemdragdrop

Boolean. Default false. If true, and Omnis is running on the Windows platform, Omnis logs the clipboard formats being dragged from the operating system and dropped on

Omnis, to the trace log. This can be useful when debugging operating system drag and drop.

macOS

The items in the **macOS** group configure behavior when Omnis is running on macOS.

allowStopInRuntime

Boolean. Default true. If true, the Cmnd+. (Cmnd plus period) key press will stop execution (e.g. break a loop) in a runtime on macOS.

dragIconBackground

Boolean. Default true. If true, a fat client drag icon has a background. A background is typically required because a themed SVG icon can appear almost invisible if dragged without a background. You can set this to false if you want your drag icons to appear as they do in Omnis Studio 10.2.

macOSbuttonNewTextDrawingStyle

Boolean. Default true. Controls the text color used for text on fat client system buttons (buttons with $buttonstyle set to kSystemButton).

If true, text on default buttons draws using the color colorpushdefaulttextmacos in the pushButton section of appearance.json; and text on pressed buttons draws using the color colorpushpressedtextmacos.

If false, text on default and pressed buttons draws using their text color.

macOStreeOutlineStyle

Boolean. Default true. If true, text in the fat client tree control draws using the colors coloroutlinetextselected and coloroutlinetextdeselected in the system section of appearance.json.

If false, text in the fat client tree control draws using the colors colorhighlighttext and colorwindowtext in the system section of appearance.json.

menuTrackingSuppressTimers

Boolean. Default true. If true, timers do not fire during menu tracking; any timers that would fire during menu tracking are deferred until menu tracking completes. This is the preferred setting.

If false, timers can fire during menu tracking.

monitorDockKeyEvents

Boolean. Default true. Set this to false to prevent Omnis from monitoring keyboard events from the Dock; this also prevents the system dialog requesting permission to do this from being shown.

odbcdam.ini

Use this character string to configure the environment variables for the ODBC DAM. A comma-separated list of entries of the form var=value.

oracle8dam.ini

Use this character string to configure the environment variables for the Oracle DAM. A comma-separated list of entries of the form var=value.

postKeyEvents

Boolean. Default true. When true this allows the Queue keyboard event command to post key events which use a modifier key.

This should be true to send the correct modifier key for the current keyboard using Queue keyboard event. To allow this key event to be sent the system will display a one-time prompt requesting that the user allows accessibility access for events.

If this is false the prompt will not be shown and a key will be sent without applying the modifier.

preventUpdateWithNoRefreshOn

Boolean. Default true (for Studio 11 onwards, false in previous versions). When set to true and a window has set $norefresh to kTrue then this will prevent changes to the window hierarchy, e.g. adding fields, from causing a redraw to screen. The window changes will then be applied when $norefresh is set to kFalse.

stackSize

Integer. Default 1048576. The stack size in bytes for a new thread created by Omnis. Set this item to zero to use the operating system default.

sybasedam.ini

Use this character string to configure the environment variables for the Sybase DAM. A comma-separated list of entries of the form var=value.

useDictation

Boolean. Default true. Specifies whether Omnis supports the operating system dictation interface.

useFnInMenuShortcuts

Boolean. Default true. If true, menu shortcuts for function keys display as Fn. If false, menu shortcuts for function keys display as ⌘n.

useToolbarStyleExpanded

Boolean. Default false. Only applies to macOS 11 and later. When true, the window toolbar style is the legacy expanded style, i.e. toolbars sit under the window title. When false, the window toolbar style is the standard macOS 11 and later style, i.e. toolbars are unified and to the right of the window title.

methodEditor

The items in the **methodEditor** group configure the behavior of the Method Editor (including the Code Editor).

badNotationNameIsSyntaxError

Boolean. Default true. If true, the method editor treats a bad notation name as a syntax error rather than automatically escaping it using a pair of double slashes.

checkFileClassPrefixBasedOnUniqueFieldNames

Boolean. Default true. Note that true is the recommended setting.

When $uniquefieldnames for the library is true, and **checkFileClassPrefixBasedOnUniqueFieldNames** is true, it prevents you from entering a file class name prefix (for file classes in the same library as the class being edited).

When $uniquefieldnames for the library, and **checkFileClassPrefixBasedOnUniqueFieldNames** is true, it requires that you enter a file class name prefix (for file classes in the same library as the class being edited).

extraLineSpacing

Integer. Default 1 pixel. Extra spacing added to the height of each code line.

maxHeightOfMethodTooltipGeneralInformation

Integer. Default 100 pixels. The maximum height of the general information section displayed at the top of a method tooltip. If the general information needs more than this space, it is displayed with a vertical scrollbar.

maxVisibleMethodLinesInMethodTooltip

Integer. Default 20 lines. The maximum number of visible text lines displayed in the code section of a method tooltip. If the method needs more space, it is displayed with a vertical scrollbar.

modifyMethodsCommandSetsFocusToTree

Boolean. Default false. This affects the focused field in the method editor after opening the method editor using a modify command of some sort (e.g. by opening the methods using the Studio browser) when the method editor for the class is already open. False means the focused field is not changed. True means the focus is moved to the method tree.

printMethodsWithSyntaxColors

Boolean. Default true. Specifies whether method text is printed with syntax coloring. Note that when using syntax coloring, the colors used are those from the default appearance theme.

selectAllCanScrollCodeEntryField

Boolean. Default false. Specifies whether select all in the code entry field scrolls the field to the end of the method rather than leaving the scroll position unchanged.

stripTrailingEmptyCommands

Boolean. Default true. Specifies whether the code editor removes trailing empty lines from a method before saving it back to the class.

useDefaultColorsForClipboard

Boolean. Default true. Specifies the appearance theme used for syntax colors when placing method text on the clipboard, for the HTML (and RTF on macOS) formats.

If true, use the default appearance theme, or if false use the current appearance theme.

validateEventsForOnCommand

Boolean. Default true. Specifies whether the code editor validates events entered for the On command.

When validation is enabled, the code editor checks to see if the code is valid for the object: if not, it flags it as an error. If the event is valid, and the method is in a remote form, and the event is not specified in $events for the object, the editor automatically adds it to $events when editing a method named $event in a non-inherited object. The editor displays a temporary status bar message after it does this.

methodeditorandremotedebugger

includeObjectNodesInTreeSearch

Boolean. Default true. If true, searching the method tree searches object nodes in addition to searching method nodes.

obrowser

The items in the **obrowser** group configure the behavior of the obrowser fat client external component (CEF - Chromium Embedded Framework).

cefSwitches

An array of character strings. Each string in the array is a switch passed to CEF when initialising CEF, e.g. --disable-logging

clearCacheWhenLoaded

Boolean. Default true. If true, obrowser clears the cache just after Omnis loads obrowser.

clearLocalStorageWhenClearingCache

Boolean. Default false. If true, obrowser clears local storage just after Omnis loads obrowser. This occurs just after checking and clearing the cached based on the value of **clearCacheWhenLoaded**, but note that local storage can be cleared (depending on the value of **clearLocalStorageWhenClearingCache**) even if **clearCacheWhenLoaded** is false.

defaultHtmlcontrolsFolderInDataFolder

Boolean. Default false. If **htmlcontrolsFolder** is empty, specifies if the default HTML controls folder is in the Omnis program folder or the Omnis data folder.

htmlcontrolsFolder

The pathname of the folder in which any HTML controls are located. If this is empty, HTML controls are located in the folder named htmlcontrols, immediately subordinate to the Omnis program or data folder, depending on the value of **defaultHtmlcontrolsFolderInDataFolder**.

locale

The locale to be used for CEF. Defaults to empty, meaning use the locale of the Omnis program (as returned by the Omnis locale() function). If not empty, it must be a locale string that can be used to set the locale of CEF, e.g. it_IT. This affects for example the context menus displayed for HTML pages rendered by obrowser.

logSeverity

Integer. Default 99. The CEF log severity. Valid values are 99 (disabled), 0 (default: info), 1 (verbose), 2 (info), 3 (warning) or 4 (error).

messageTimeout

Integer. Default 60. Used when obrowser is communicating via a local websocket with HTML controls. The timeout in 1/60th second units after which obrowser stops waiting for an expected response from the HTML control.

remoteDebuggingPort

Integer. Default 5989. A value between 1024 and 65535. The port number passed to CEF as its remote debugging port. You can debug pages attached to obrowser instances by opening a web browser and navigating to the URL http://localhost:<**remoteDebuggingPort**>.

useOmnisTraceLogForConsole

Boolean. Default true. If true, console messages from pages hosted by obrowser are redirected to the Omnis trace log.

ocx

The items in the **ocx** group configure the behavior of the OCX external component.

markPtrAsAltered

Boolean. Default true. Provides the ability to handle issues when using functions for certain automation objects that have parameters of type VT_PTR; setting this to false can work around some of these issues. However, you should keep this option set to true unless you are advised otherwise by Omnis Support.

ole2auto

The items in the **ole2auto** group to configure the behavior of Windows platform Automation.

enableEvents

Boolean. Default true. Some automation servers fire events, for example, an email application may fire an email alert event to signify new emails. To enable these events set this item to true.

omnishttpserver

The items in the **omnishttpserver** group configure the behavior of the built-in Omnis HTTP server. This server is primarily intended to be used for the local testing of remote forms.

preventclientcaching

Boolean. Default true. If true, the server adds headers to its responses that tell the client that it must not cache the returned content.

pdf

The items in the **pdf** group configure the behavior of the Omnis PDF device.

On the Windows platform only, you can specify font name mappings as additional items within this group. These items have an entry name which is the name of the font (as used in Omnis) and an entry value which is the name of the font as it appears in the Windows registry.

omnispdfFolder

Used to configure the folder in which the Omnis PDF device generates scripts and PNG files. Defaults to an empty string (meaning use the omnispdf folder in the Omnis data folder). If not empty, it must be the path of the folder to use in place of omnispdf. The folder must already exist (otherwise Omnis reverts to the normal omnispdf folder).

plainSuffixes

Windows platform only. A comma-separated string of suffix values that can be used to identify fonts with a plain style. This is needed because different languages name these fonts using different suffixes. The string defaults to "Regular,Standard,Normal,Normale".

properties

The items in the **properties** group configure the behavior of the Property Manager.

show_editor

Boolean. Default false. If true, the property manager includes the properties $editor and $editordata where relevant.

server

The items in the **server** group configure the behavior of the Omnis Server.

bindAttempts

Integer. Default 5 in a dev version, 0x7fffffff in a server or runtime version. The maximum number of attempts (1 second apart) to bind to the Omnis server port.

disableInRuntime

Boolean. Default false. Only applies to a runtime (or server) version. If true, Omnis does not try to bind to the Omnis server port. Useful if you do not want the server to be active, and you want to prevent operating system firewall prompts.

getpdfFolders

The JavaScript client getpdf command only allows non-temporary PDF files to be retrieved from a fixed set of folders. You specify these folders using this item, which is an array of folder paths. Each entry is a folder path, without a trailing file separator.

headlessAcceptConsoleCommands

Boolean. Default true. Applies to headless Linux server only. If true, and the headless server is run synchronously from a console or terminal, you can use the keyboard to enter commands, for example to quit the headless server. To see a list of possible commands, press return.

headlessDatabaseLocation

The pathname of the SQLite database file used by the headless server admin tool (osadmin). If empty, the admin tool uses the file osadmin.db in the same directory as the osadmin library. If the database file does not exist, osadmin creates a new database file.

iconsFolderName

Allows you to override the name of the icons folder in the html folder in the Omnis data folder. If this item is empty or omitted the icons folder is html/icons;
otherwise it is html/<**iconsFolderName**>.

multiProcess

Boolean. Default false. Applies to headless Linux server only. If true, the headless server runs in multi process mode.

overridePushURL

Use this to override the default push URL generated by the JavaScript client scripts: this can only be used when using openpush in a server method. To use the default push URL, set this item to empty.

port

A character string. The TCP/IP port number (1-32767), or service name, on which the Omnis Server listens for connections. This is where $prefs.$serverport is stored.

readTimeout

Integer. Default 20. The length of the read timeout in seconds. Only applies if **timeoutReads** is true.

RESTfulConnection

[POOL,][IPADDR:][PORT]. Controls how the Omnis RESTful Web Server plugin connects to Omnis. POOL is a load sharing process pool name; IPADDR and PORT identify Omnis or load sharing process; if empty,defaults to $serverport. This is where $prefs.$restfulconnection is stored.

RESTfulURL

The base URL used to call Omnis RESTful Web Services, e.g. http://www.test.com/scripts/omnisrestisapi.dll. Omnis uses this in the OpenAPI and Swagger definitions it generates. If empty,Omnis uses http://127.0.0.1:$serverport. This is where $prefs.$restfulurl is stored.

retryBind

Boolean. Default true. If true, and binding to the server port fails, Omnis will retry binding (once a second) for up to **bindAttempts** times.

runtimeOpensTraceLogOnSocketBindError

Boolean. Default true. If Omnis fails to bind to the Omnis server port, after retrying if allowed, it writes an error message to the trace log. If this item is true, then a runtime or server version of Omnis also opens the trace log after writing the error message.

showBindRetryMessage

Boolean. Default true. If true, Omnis displays a working message while it is retrying the bind to the Omnis server port.

stacks

Integer. Default 5. The number of threaded stacks set up by the Start server command. Must be between 1 and 99 inclusive. This is where $prefs.$serverstacks is stored.

start

Boolean. Default false. This setting is ignored by the Linux headless server if multiProcess is set to true. Otherwise, set this to true to automatically execute *Start server* when Omnis starts.

timeOffsetMinutes

Integer. Default zero. Omnis adds this to the current system date-time when generating the value for #D and #T. This allows the server date and time evaluated in Omnis code to be adjusted to match the time zone of its clients.

timeoutReads

Boolean. Default true. If true, the Omnis server times out and closes inactive connections from clients. A connection is considered inactive if either no message has been received, or only a partial message has been received, and no further data is received in **readTimeout** seconds.

timeslice

The duration (in 1/60th second units) of the execution time slice for a server thread. A value of less than or equal to zero results in the timeslice value actually used being 20. This is where $prefs.$timeslice is stored.

webServiceConnection

[POOL,][IPADDR:][PORT]. Controls how the WSDL Web Service Web Server plugin connects to Omnis. POOL is a load sharing process pool name; IPADDR and PORT identify Omnis or load sharing process; if empty,defaults to 127.0.0.1:$prefs.$serverport. This is where $prefs.$webserviceconnection is stored.

webServiceLogging

A string that specifies how the Omnis WSDL Web Service Server logs requests:off for no logging, faults to log SOAP faults only, full to log all requests. This is where $prefs.$webservicelogging is stored. See also the library preference $disablewebservicelogging.

webServiceLogMaxRecords

The maximum number of records allowed in the WSDL Web Service request log; once reached,the server deletes the oldest record before inserting a new record. Setting a new value deletes records if the new limit is exceeded. This is where $prefs.$webservicelogmaxrecords is stored.

webServiceStrictWSDL

Boolean. Default true. If true, Web Service WSDLs are strict (types are qualified with schema restrictions where possible, and annotations can be added to schema types). Set this to false if your client objects to this additional information in the WSDL. This is where $prefs.$webservicestrictwsdl is stored.

webServiceURL

The URL used to call Omnis WSDL Web Services, e.g. http://www.test.com/scripts/owsisapi.dll. Omnis uses this in the WSDL files it generates. If empty, Omnis writes http://127.0.0.1:$serverport to WSDL files (use empty for local testing). This is where $prefs.$webserviceurl is stored.

servermgmt

The items in the **servermgmt** group configure the behavior of the Omnis Server Management Library (servermgmt.lbs).

showTrayIcon

Boolean. Default false. Controls whether servermgmt.lbs shows the tray icon.

svg

The items in the **svg** group configure the behavior of the SVG icon support.

customSizes

An array that allows you to configure the custom sizes available in the IDE select icon dialog, when specifying the dimensions of an SVG icon. Note that the select icon dialog also allows you to manipulate this array. Each entry is of the form wxh, specifying the width and height of the custom size e.g. 256x256.

tooltips

The items in the **tooltips** group configure the behavior of tooltips.

generalDisplayDelay

Integer. Default 500. The delay in milliseconds before a general (non-method) tooltip displays while the mouse is held over an object that has a tooltip.

maxWidth

Integer. Default 0. The maximum pixel width of a tooltip. A value less than or equal to zero means use a third of the screen width. This item applies in all but a few special cases.

methodContentDisplayDelay

Integer. Default 500. The delay in milliseconds before a method content tooltip displays while the mouse is held over an object that has a method tooltip.

vcs

The items in the **vcs** group configure the behavior of the Omnis VCS.

enableBranching

Boolean. Default false. If true, the Omnis VCS supports branching.

web

The items in the **web** group configure the behavior of the Omnis WEB method commands.

ftpsresumesession

Boolean. Default true. If true, the FTP commands attempt to resume the control connection session when establishing a secure data connection.

windows

The items in the **windows** group configure behavior when Omnis is running on Windows.

backgroundObjectsMustUseTrueTypeFont

Boolean. Default true. Applies to fat client label and text background objects. If true, these require a TrueType font; if false, they will draw using any font. If you set this item to false, there may be certain cases e.g. when dragging the objects in design mode, where the text does not draw.

canConvertAnsiPageSetupData

Boolean. Default true. If true, Omnis converts Ansi page setup data in converted non-Unicode report classes to Unicode.

createShortcut

Boolean. Default true. If true, and there is no shortcut to itself in the Start menu, Omnis creates a shortcut to itself in the Start menu. It then modifies the shortcut to contain the AppUserModelID required for local notifications to work.

defaultMenuDesignDPI

The default scaling DPI values used for menus on the Windows platform when **defaultMenuDesignDPIMode** is not set to kDPIoff. A comma separated string of three DPI values. Default 96,72,75.

defaultMenuDesignDPIMode

A character string specifying the DPI mode for menus on the Windows platform. Default kDPIoff. Either kDPIoff, kDPIframeOnly or kDPIall.

forceHighDPIawareMode

Integer. Default zero. Set this to be 1, to force Omnis into high DPI mode, irrespective of the DPI (only recommended on a retina screen or higher; high DPI mode will activated and DPI set internally to 192).

hideStudiorgMessage

Boolean. Default false. If true, it prevents Omnis from displaying the dialog informing the user that studiorg will run.

highDPIaware

Boolean. Default true. If true, Omnis will tell Windows that it is DPI aware, meaning that on retina resolution screens it will operate at high DPI (192).

includeDDEEditMenuItems

Boolean. Default false. If true, the DDE menu items "Paste Link" and "Remove DDE Link" are added to the Edit menu. Omnis requires a restart after editing this item.

initLocalNotifications

Boolean. Default true. If true, Omnis initialises the interfaces required to send notifications to the local Notification Center. See also the createShortcut configuration entry in the windows section.

miniconid

The icon id of the minimize icon for the Omnis executable.

noAdmin

Boolean. Default false. When true, Omnis does not try to run studiorg with administrator privileges.

pythonPath

If the Omnis PDF device is using python to generate PDF reports, this item allows you to override and specify the path of the python executable used to run the PDF generator python scripts. Leave this empty to use the default python executable in the Omnis tree.

readBorderActiveColorFromSystem

Boolean. Default true. If true, Omnis reads the accent color from the system, and uses that as the color for active window borders, when $prefs.$windowoptions specifies Windows 8, 8.1 or 10, and borderactivecolor in $prefs.$windowoptions is set to kColorDefault. If false, Omnis uses a hard-coded color (rgb(244,112,35)) for borderactivecolor when borderactivecolor is kColorDefault.

scaleScreenCoordsUsingPhysicalSize

Boolean. Default false. Defaults to false. Only applies when $clib.$screencoordinates is true. If false, screen coordinate scaling is based on the size of the main window; if true, it is based on the physical screen size.

singleInstance

Boolean. Default false. Controls the default value of $prefs.$singleinstance. On Windows, if the $singleinstance root preference is set to kTrue, the same instance of Omnis is used to open a library file, otherwise another instance of Omnis will be started.

updateFileAssociations

Boolean. Default true. If true, Omnis will if necessary attempt to associate its file extensions (e.g. .lbs, .df1) with the current running executable, by executing studiorg at startup.

useLegacyDefaultPrinter

Boolean. Default false. If false, Omnis uses the Windows GetDefaultPrinter API to obtain the default printer. If true (present for potential backwards compatibility issues) Omnis uses the Win16-compatible API GetProfileString).