What’s New in Omnis Studio 8.1.7

Omnis Software
April 2019
48-042019-01a

Table of Contents

About This Manual

This document describes the enhancements in Omnis Studio 8.1.7, plus all 8.1.x & 8.0.x releases, as well as 8.0, including many new features in the JavaScript Client and in the Studio IDE.

Please see the Readme.txt file for details of bug fixes and any last-minute notes for the 8.1.x release. See the Install.txt file for information about installation.

If you are new to Omnis Studio

When you start Omnis Studio for the first time the **Welcome** window will be displayed, which provides a short interactive tutorial, or walkthrough, that allows you to create a simple app that you can open on your desktop or web browser. The **Advanced** option lets you skip the tutorial and go straight to the Omnis Studio IDE.

After you close the Welcome window, you will see the **Studio Browser** which provides access to all the main tools in Omnis Studio (if this is not showing press F2, or click on the Compass icon marked 'Browser' on the main Omnis toolbar). The **Hub** should be selected in the Studio Browser which provides information and videos to help you get started in Omnis Studio, as well as information about recent reported faults in Omnis Studio. You can look at some example Omnis applications under the **Applets** and **Samples** options: you can open each example in your web browser or within the Omnis IDE, and you can examine the Omnis code in the associated library under the **Libraries** option in the Studio Browser.

Creating a new library

To create a new Omnis application (library) you can click on the **Libraries** option in the Studio Browser, click on the **New Library** option and step through the process of creating an Omnis application using the **App Builder.** This provides a number of different options for starting your application, including from a sample database, your own database, or by importing some data from a comma- or tab-separated file.

Software Support and Compatibility

Serial Numbers and Licensing

If you are upgrading from any previous version of Omnis Studio, including 8.0.x, 6.1.x, 6.0.x or 5.x (and before), you will require a new serial number to run the Development version of Omnis Studio 8.1.x. Contact your local sales office for further details about new development and deployment licenses for Omnis Studio 8.1.x.

JavaScript Client App Server Licensing

Multiple connections to the JavaScript Client App Server from a single client browser are now counted as only one use of a Server license. In versions prior to Studio 8.1.x, multiple connections from a single client browser were being counted as separate users and consuming server licenses.

When the JavaScript Client communicates with a web server, it generates a UUID to identify itself, and (from Studio 8.1.6) saves it in localStorage which it sends as a parameter whenever it connects to the server. (Prior to this version, the UUID was stored in a cookie which required any clients to have cookies to be enabled for this licensing mechanism to work.)

Web Services Serial Number

You no longer require a Web Services serial number to use the REST based web services feature in the Professional Edition of Omnis Studio 8.1.x or above. However, in order to use the HTTPClientWorker object to create a Web Services client you still need to install and configure Java (not required for the new OW3 CURL based worker objects). If you are creating your own Web Services, from your Omnis code, your server will still require a deployment Omnis App Server license when you are ready to deploy your app.

Runtime Maintenance Agreements

With Omnis Studio 8.0.x or above we have introduced a new Runtime Maintenance Agreement (RMA). For further details about these new agreements, please contact your local sales office.

Library and Datafile Conversion

Omnis Studio 8.1.x will convert existing version 8.0.x, 6.1.x, 6.0.x and 5.x libraries – THE CONVERSION PROCESS IS IRREVERSIBLE.

Omnis Studio 8.1.x will convert version 5.x Omnis datafiles (note that non-Unicode datafiles will be converted to Unicode), but 8.0.x, 6.1.x and 6.0.x datafiles *will not* be converted in Omnis Studio 8.1.x.

IN ALL CASES, YOU SHOULD MAKE A SECURE BACKUP OF ALL LIBRARIES AND OMNIS DATAFILES BEFORE OPENING THEM IN OMNIS STUDIO 8.1.X.

NOTE: The Omnis Datafiles browser is not displayed in the Studio Browser when you first start Omnis Studio 8.1.x: to display the Omnis Datafiles browser, you need to enable it under the new **Options** setting under the Hub option in the Studio Browser.

macOS Support and Version Check

Omnis Studio 8.1.x 64-bit is certified to run against a minimum of macOS version 10.11 (El Capitan) or higher. Omnis may continue to run on 10.9 and 10.10, but these are not supported and any issues specific to these versions of the OS will not be addressed.

The Omnis Studio 64-bit application will check the version of macOS and will not run if it is older than the minimum requirement to run Omnis Studio on macOS. In this case, Omnis Studio will be marked with a disabled icon.

Renaming OS X to macOS

With the release of macOS Sierra, Apple renamed “OS X” to “macOS”, therefore we renamed all occurrences of “OS X” and “OSX” to “macOS” in Omnis Studio (starting with 8.0.3 and above). The changes in Omnis Studio are mainly in the notation, such as property names and descriptions, theme colors for $appearance, and in the online docs, as well as the Omnis Help. For example, all $osx… properties have been renamed to $macos…, and some constants have been renamed, such as kMacOSX to kmacOS.

This change should not affect the majority of your code since this is a straightforward update in the string resources in Omnis Studio, but you should check your libraries for any literal occurrences of “OS X” and similar usage and update those accordingly.

Sync Server

A new version of the Sync Server, version 2.3, which was released with Studio 8.0.3, uses a RESTful interface to allow the Omnis Server to communicate with mobile clients: note new wrappers are required for this version of Sync Server. See the Sync Server section below for further details.

Java 8

To use Java in Omnis Studio 8.0.x or later for development and deployment (such as Java Objects) you now need to install and reference **Java Version 8,** which is available from Oracle: you can download the Java Developer Kit (JDK), for Windows or macOS, or Java Runtime Environment (JRE), for Windows only, from the following location:

- http://www.oracle.com/technetwork/java/javase/downloads

Java Configuration

Having installed the latest JDK or JRE you need to configure the JVM, either using a new entry in the Omnis configuration file (config.json), or by setting an environment variable: OMNISJVM64 or OMNISJVM32, depending on whether you are running the 64-bit or 32-bit version of Omnis Studio. If you specifiy a value in config.json, it overrides the value in the environment variable.

To setup the JVM in the config.json file, update the “jvm” entry in the “java” object in the configuration file, for example, on Windows:

```
"java": {
     "jvm":"c:\\Program Files\\Java\\jre8\\bin\\server\\jvm.dll",
     "resetClassCacheOnStartup": false
      }
```

Or on macOS:

```
"java": {
     "jvm":"/Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/
   Contents/Home/jre/lib/server/libjvm.dylib",
     "resetClassCacheOnStartup": false
      }
```

You can set the JVM in the config.json file on a Linux server in a similar manner.

Web Services

Support for REST based Web Services was introduced in Omnis Studio 6.1, including support for **Swagger** definitions to define an Omnis RESTful API for creating your own web services from Omnis code. From Studio 8.0 onwards, Omnis supports Swagger 2.0 rather than 1.2 for RESTful web services. This only affects the Swagger files Omnis generates, and there is now just one definition per service. There is a ‘Save to File’ link to save the Swagger file for a service to disk under the Web Service Server node in the Studio Browser.

In addition, the defaultreslist.json file has been replaced with a file called default.json (in the same location). The nickname property (in both the method editor and notation) has been replaced with operationid, therefore $httpoperationid replaces $httpnickname.

Omnis uses the first non-empty description it can find for a remote task in the service as the description of the service in the Swagger file.

OpenSSL

There are a number of Web commands that relied on OpenSSL in previous versions to provide secure communications: these included FTPConnect, HTTPOpen, TCPConnect, POP3Connect, and so on. We have removed the reliance on OpenSSL which means you no longer have to install it to use secure connections in these commands. Instead, the built-in security technology will be used, so on Windows 'Secure Channel' (Schannel) is used, on macOS 'Secure Transport' is used, and on Linux OpenSSL will continue to be used since it is the default security technology on Linux. See the Omnis Help for details about these commands (FTPConnect, etc).

Welcome and New Users

When you first launch Studio 8.1 a **Welcome** window will open allowing you to create a small "Hello World" type application containing a JavaScript remote form or a window class. The Advanced otion will take you straight to the Omnis IDE, or if you don't want the Welcome window to appear again, you can uncheck the 'Show at Startup' option.

The Applets and Samples that used to appear in the old Welcome screen have been added to the **Hub** in the Studio Browser. The tutorial is available in Chapter 1 of the Web Development manual on the Omnis website, including a ZIP file of the Tutorial example libraries.

In addition, a new tool called the App Builder, that allows new users to create or prototype applications quickly and easily, has been added to the 'New Library' option under the Libraries option in the Studio Browser.

CPU type: sys(110)

The sys(110) function returns the CPU type of the computer running Omnis. This function is no longer supported in Studio 8.0.x or higher, on the 64-bit versions of Omnis Studio, on all platforms. Since there is such a variation of processor types, across all types of computers and devices, this function is no longer a reliable indicator of the computer type or platform Omnis is running on.

External Components

The external components Flic, MCIplay, NPAPI, Pcx, Stix and Wbmp are no longer supported in this release and have been removed from the Xcomp directory for all platforms.

Picture Formats

The WBMP and PCX picture formats are no longer supported in this release. This affects the pictconvto/from functions and any other places where you set the picture format.

FileOps Functions

The $readentirefile and $writeentirefile FileOps functions are no longer supported and have been removed from the Functions tab in the Catalog.

VCS Branching

Access to branching in the VCS has been removed from the Studio Browser, although VCS branching will continue to work in existing projects for backwards compatibility: see the section on VCS for details.

Mac Touch Bar

The Mac Touch Bar API can be enabled/disabled via plist; it is disabled by default. There is a new Boolean entry called "enableTouchBar" in the Info.plist inside the Mac application package. To enable the touch bar set this to YES.

Windows Path names

Under Windows, if you use paths of 240 characters in length or longer, you must use \ (back slash) as the separator in such paths. For paths under 240 characters you can use either \ or / in paths. This applies for all commands and functions in Omnis that require a path as a parameter.

PDF Font Mapping

When using custom fonts for PDF printing there may be a mis-match between the name of a font and its Window registry entry, which results in the font not being found and the report not being rendered correctly. To rectify this, you can add mappings to the "pdf" entry in config.json (that apply to the Windows platform only), to map a font name to its entry in the registry. For example, you can map the font name "Proxima Nova Rg" to its registry entry "Proxima Nova Regular", using the following item in the config.json file.

```
"pdf": {
   "plainSuffixes": "Regular,Standard,Normal,Normale",
   "Proxima Nova Rg": "Proxima Nova Regular",
   "Proxima Nova Rg Bold": "Proxima Nova Bold"
},
```

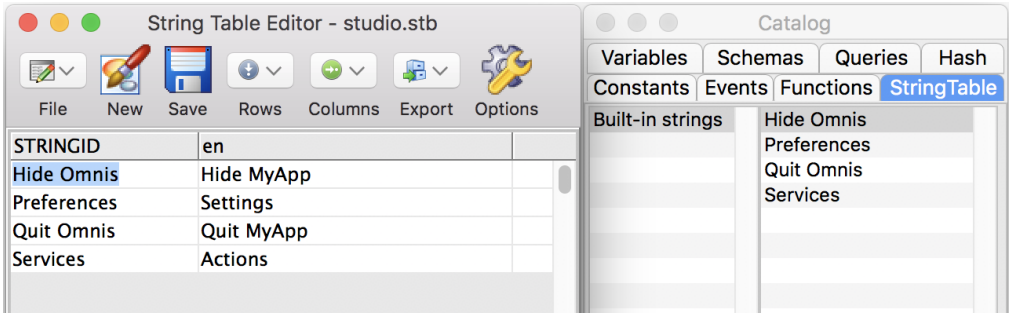
What's New in Omnis Studio 8.1.7

Omnis Studio 8.1.7 contains the following minor enhancements, and some fault fixes which are listed in the Readme text file available with the download.

Localization

Changing System menu items (macOS)

You can change the **Hide Omnis** and **Quit Omnis** options in the Omnis Studio runtime on macOS by adding strings to the Studio String Table (studio.stb). You can now localize items in the **Preferences** and **Services** menus. Note you can find specific strings in Omnis Studio using the Find strings… option by right-clicking on the string table name in the Catalog.



JavaScript Components

JavaScript Component Templates

When you add a JavaScript Component to a remote form in your code at runtime, Omnis now uses a template to create the object with all the required properties and methods. There is a template for every type of JavaScript Component, and the templates are located in the \studio\componenttemplates folder.

The component templates match the default components in the Component Store, and should not be edited. There are templates for Report and Window class components as well.

Omnis Programming

Maximum Number of Methods

The maximum number of methods per class has been increased to 4096 from 501. (ST/PF/622)

What's New in Omnis Studio 8.1.6

Omnis Studio 8.1.6 contains the following minor enhancements, and some fault fixes which are listed in the Readme text file available with the download.

OBrowser

A number of enhancements have been made to the OBrowser window component, to enable the use a new HTML control JSCBridge that allows you to run the Omnis JavaScript Client within OBrowser in a desktop (fat client) form, and to enhance HTML controls including support for dates.

JavaScript Client Bridge

The **JavaScript Client Bridge** (JSCBridge) is a new HTML control that allows you to run the Omnis JavaScript Client within OBrowser in a standard Window Class, which means you can open a Remote Form in the desktop (fat client) version of Omnis Studio, passing data between the form and Omnis.

The JSCBridge control source code and documentation is available on the Omnis Studio GitHub repository:

https://github.com/OmnisStudio/Omnis-JSCBridge

HTML Controls & Dates

The OBrowser window class component allows you to embed HTML controls on a window class.

HTML controls can now use *Date variables* as their $dataname. In order to use dates, the .htm file for the control needs to include a link to a new JavaScript file **omn_date.js** which enables dates: this has been added to the template included in this version, but you will need to add it to any existing html for controls.

HTML controls can now also pass dates when calling sendControlEvent(), either directly, or as a column in a row/JS object. In addition, HTML controls can pass nested rows/JS objects with sendControlEvent().

The omnisOnWebSocketOpened() method now receives the Web Socket port as a parameter.

For more information about creating HTML controls for desktop (fat client) apps, refer to the *Omnis Programming* guide on the Omnis website.

oBrowser & localStorage on macOS

oBrowser on macOS now overrides the WebView's handling for localStorage for file URLs (only). It writes the localStorage keys & values to a file called localStorage.json in the clientserver/client/ folder.

JavaScript Components

Control Classnames

All JavaScript controls now have a base class name to allow you to control the appearance of controls using CSS, and apply a consistent appearance for each type of JavaScript control. The classnames listed below can be added to the 'user.css' and CSS properties applied to the classname to control the appearance of each type of

control. Note these classnames are contained in the JavaScript controls by default and if they are added to the user.css are applied to the control automatically, that is, the new classnames do not need to be included in the $cssclassname property to be applied. You can add further styles to user.css and quote them in $cssclassname to apply those additional style properties to individual controls (as in previous versions).

JS Control	Class Name	Additional notes
'Frame' element for all controls	omnis-[control]-frame	
Activity Control	omnis-activity	
Background Control	omnis-background	
BarChart Control	omnis-barchart	
Button Control	omnis-button	
Checkbox Control	omnis-checkbox	
ComboBox Control	omnis-combo	The dropped list has "ctrl-drop-list" assigned. If ($cssclassname) the opened items list will be assigned the class of the first class in $cssclassname suffixed with "-dropped-list"
Complex Grid	omnis-complexgrid	omnis-complexgrid-header and omnis-complexgrid-hheader for header and horizontal header areas. Each row has omnis-complexgrid-row and either 'odd' or 'even' depending on their line number. If ($cssclassname) the header/hheader will have class $cssclassname+"-header" and "-hheader"
Date Picker Control	omnis-date	
Data Grid Control	omnis-datagrid	
Droplist Control	omnis-droplist	The dropped list has "ctrl-drop-list" assigned. If ($cssclassname) the opened items list will be assigned the class of the first class in $cssclassname+"-dropped-list"
Edit Control	omnis-input	
File Control	omnis-file	
HTML Object	omnis-html	
Hyperlink Control	omnis-hyper	

JS Control	Class Name	Additional notes
Label Object	omnis-label	
List Control	omnis-list	
Map Control	omnis-map	
Menu - used for context menus, popup menus and tab menus	omnis-menu	omnis-menu-main for containing omnis-menu-table for table omnis-menu-row for row omnis-menu-cellcheck for check or icon element in the menu omnis-menu-celltext for the text element omnis-menu-cellcascade for the cascading menu element Popup and tab menus will implement If ($cssclassname) the opened items list will be assigned the class of the first class in $cssclassname+"-opened-menu"
Native List Control	omnis-nativelist	
Native Slider Control	omnis-nativeslider	
Native Switch Control	omnis-nativeswitch	
Navigation Bar Control	omnis-navbar	
Navigation Menu Object	omnis-navmenu	
Page Control	omnis-pagectl	
Paged Pane	omnis-pagedpane	
Picture Control	omnis-picture	
Popup Menu Control	omnis-popup	Also contains the classes from omnis-menu as it uses this object for the menu element of the control.
PieChart Control	omnis-piechart	
Progress Bar Control	omnis-progress	
RadioGroup Control	omnis-radio	
Rich Text Editor Control	omnis-rich	
Segmented Control	omnis-segmented	
Slider Control	omnis-slider	
Subform	omnis-subform	
Switch Control	omnis-switch	

JS Control	Class Name	Additional notes
Tab Control	omnis-tabs	Also contains the classes from omnis-menu as it uses this object for the menu element of the control.
TransButton Control	omnis-trans	omnis-trans-text To address text element of a trans button.
Video Control	omnis-video	

For example, to add CSS styling to all the Edit controls in your remote forms you could add the following CSS to the user.css file in the 'html/css' folder in the main Omnis folder: in this case, the base classname .omnis-input is used with the properties 2px solid grey border and a 6px radius.

```
.omnis-input {
    border: 2px solid grey;
    border-radius: 6px;
}
```

JavaScript Remote Forms

evLayoutChanged & pBreakpoint

In previous versions the pBreakpoint parameter was reported as a string when evLayoutChanged was detected in a client executed method; this is now reported as an integer value, which matches the behaviour of evLayoutChanged in server methods. This may affect existing libraries which compare pBreakpoint with a string.

Libraries

Library Conversion

When you try to open a library that needs to be converted, Omnis will prompt you to ask if you want to convert the library: this applies to libraries you open in the development version of Omnis and any libraries located in the Startup folder that are loaded automatically.

There is a new option to suppress conversion prompts when a library that needs to be converted is opened from the Startup folder. The new option "disableAllLibraryConversionPrompts" has been added to "defaults" section of the Omnis Configuration file (config.json). The default is false, so set this option to true to prevent library conversion prompts.

Headless Server

Running as a Service

The "headlessAcceptConsoleCommands" setting in the Server group of the Omnis configuration template file (config.json) has been changed to false. Previously this was set to True which meant that all Console Commands were recorded which meant that 100% of CPU was used when the Headless server was run as a service: this new option ensures this is turned off by default, but will need to be enabled to accept all Console Commands.

SQL Programming

OmnisSQL & National Fields

In previous 8.1.x versions there was an issue when using the OmnisSQL DAM and LIKE in a WHERE clause to search data in National fields. The issue has been fixed by adding a new compare mechanism, but to use the new behavior you need to enable an entry in the Omnis Configuration file (config.json), "nationalFieldCompareChars", in the "defaults" section.

This entry defaults to false, meaning the old behavior is maintained for compatibility with existing data files, so you only need to set this option to true if there is an issue in your code.

Having set this option to true, you also need to drop and rebuild the indexes from any files containing national fields.

Deployment

App Server Licensing

In order to enforce licensing for JavaScript Client based apps, the UUID of each client is logged with the Omnis App Server. Prior to this version, the UUID was stored in a cookie in the client computer which required any clients to have cookies to be enabled for this licensing mechanism to work. However, the method for storing the client UUID has changed in this version: the UUIDs are now stored in the ‘localStorage’ on each client which is now used to manage client licenses on the Omnis App Server. Therefore, clients no longer have to have cookies enabled for App Server Licensing to be enforced.

What’s New in Omnis Studio 8.1.5

Omnis Studio 8.1.5 contains the following minor enhancements, and some fault fixes which are listed in the Readme text file available with the download.

SQL Programming

$definelistorrow method

There is a new method for session objects, $definelistorrow(&vListOrRow, cTableName) which defines a list or row variable from a server table. The method returns a Boolean, true for success.

If you pass $cinst as the list or row argument from within a table class, the call maintains and updates the table instance associated with the parameter. If the server table contains a primary key, $definelistorrow sets $excludefromwhere to kTrue for non-primary key columns.

$usescale

There is a new property, $usescale (a Boolean property), for the ODBC DAM. If kTrue, Omnis number dp columns are bound using a precision of 15 + the dp value. If kFalse (the default), number dp columns are bound using a precision of 15. Also affects $createnames().

Window Programming

Edge Float Properties in Subclasses

In Studio 8.1 the behavior changed with regards to Float properties for fields and controls in Subclasses. If you open a window which has a superclass, and which overrides $width or $height, children in the superclass now float to reflect the difference in size between the superclass and the class.

There is a new entry setting, "floatWindowSubclass" (default true), in the "defaults" section in config.json, which allows you to override the new Subclass window floating behavior. You can set this to false to revert to the previous floating behaviour with subclasses (i.e. not floating if width or height are overridden).

JavaScript Components

Labels and Date variables

Label fields now support Date variables as their $dataname, taking their display format from #FDT, #FD, or #FT depending on whether they have a date/time component.

Window Components

Toolbar button text on macOS

In previous macOS versions, if the text on a toolbar button was quite long it was not displayed, while other shorter names were displayed, giving an inconsistent appearance.

In this version on macOS, toolbar buttons will resize to accommodate longer text names, and so the button names are always displayed, unless $showtext is off. This is more in line with the behavior on Windows.

Functions

sys(237)

There is a new sys() function, sys(237), which will return an item reference to the method currently being edited (in design mode), if the method editor is the top window and only one method is selected. For example:

```
Set reference item to sys(237)
If item
  OK message {[item.$fullname]}
Else
  Send to trace log no method
End If
```

What's New in Omnis Studio 8.1.4

Omnis Studio 8.1.4 contains the following minor enhancements, and some fault fixes which are listed in the Readme text file available with the download.

OW3 HTTP Workers

The OW3 HTTP worker now supports WebSocket client connections to a WebSocket server. The HTTP worker can be used to build a client interface since all WebSocket connections start off by exchanging HTTP headers that eventually upgrade the connection to a WebSocket connection.

WebSocket Server Support

$init

To initialise the OW3 HTTP worker object so that it is ready to create a WebSocket client connection, call $init with parameters as follows:

Parameter	Description
cURI	The URI of the WebSocket server, which must include the URI scheme (ws or wss) e.g. wss://demos.kaazing.com/echo You cannot omit the URI scheme, because the HTTP worker defaults to using http.
iMethod	Must be kOW3httpMethodGet
lHeaders	A two column list where each row is an HTTP header to add to the HTTP request. The worker automatically adds these headers when connecting to a WebSocket server, so do not add these headers: connection: upgrade upgrade: websocket sec-websocket-version: 13 sec-websocket-key: <key value>
vContent	Not used
iAuthType	A kOW3httpAuthType... constant that specifies the type of authentication required for this request. If you omit this and the remaining parameters, authentication defaults to kOW3httpAuthTypeNone.
cUserName	The user name to use with authentication types kOW3httpAuthTypeBasic and kOW3httpAuthTypeDigest.
cPassword	The password to use with authentication types kOW3httpAuthTypeBasic and kOW3httpAuthTypeDigest

The standard OW3 properties $state, $errortext, $errorcode, $threadcount, $protocollog, $timeout, $callprogress and $curloptions all apply when connecting to a WebSocket server.

The standard OW3 methods $getsecureoptions and $setsecureoptions apply when connecting to a WebSocket server.

$run and $start

You cannot use $run to establish a WebSocket connection, since multiple threads are required to make it usable. So if you try to use $run, the worker returns kFalse and sets $errorcode and $errortext.

To establish a WebSocket connection, call $start. If $start succeeds, then the worker attempts to establish the connection to the WebSocket server in another thread.

If the connection cannot be established, the worker generates a notification to $completed, with a non-zero value in the errorCode member of the notification row parameter.

If the connection is established successfully (and is therefore open and ready for data transfer), the worker generates a notification to the new method $ws_onconnect. Override this method to receive this notification. $ws_onconnect receives a single row variable as its parameter. This row variable has a single column, responseHeaders, which is a row with a single column for each response header received from the server in the final HTTP protocol exchange resulting in the 101 (web socket protocol handshake) HTTP status code.

As soon as you have received the $ws_onconnect notification, the WebSocket is ready to send and receive data.

$wssend

After you have received the $ws_onconnect notification, you can send data using the method $wssend:

$wssend(vMessage)

Sends the supplied message on a connected web socket. Returns true if successful, which means the message has been queued for sending.

If vMessage is a character value, the worker converts it to UTF-8 before sending it as a text message; otherwise the value is treated as binary and sent as a binary message.

Receiving Data

Each message received from the WebSocket server generates a $ws_onmessage notification. Override this method to receive these notifications. $ws_onmessage receives a single row variable parameter, with 2 columns (named data and utf8). Column data is the binary data and column utf8 is boolean true if the data is UTF-8.

$wsclose

The client can close the WebSocket connection by calling the method $wsclose:

$wsclose([bDiscardUnsentMessages=kFalse,iStatusCode=1000,cReason=''])

Closes the connection to the web socket server. $completed() will be notified when the connection has closed. The row passed to $completed has closeStatus and closeReason columns that receive the values sent in the close frame to the server.

Pass bDiscardUnsentMessages as kTrue, to discard any completely unsent queued messages before sending the close frame to the server.

iStatusCode is an integer status code that indicates the reason for closure (one of the values in section 7.4 of RFC6455).

cReason is some optional text that indicates the reason for closure.

Server close

The server can send a close frame to the client, telling the client it is closing the connection. The client responds with a close frame, before the connection closes. Again, $completed is notified to tell the object that the connection has closed.

The row passed to $completed has closeStatus and closeReason columns that receive the values sent in the close frame from the server.

$cancel

You can use $cancel to terminate the connection in a non-graceful manner.

Ping-pong

If the client receives a Ping from the server, it automatically responds with a Pong. You can also set up the client to automatically Ping the server, and generate an error (closing the connection) if a Pong is not received. To do this, use these two properties:

$wspinginterval: If non-zero, and the connection is to a web socket server, the HTTP worker sends a Ping frame to the server every $wspinginterval seconds of inactivity, and closes the connection if a Pong frame is not received after $wspongtimeout seconds. Defaults to zero.

$wspongtimeout: The number of seconds (1-60, default 5) the client waits to receive a Pong frame after sending a Ping frame as a result of the $wspinginterval timeout expiring.

Timeout

The object restarts the $timeout timer each time it sends or receives some data.

Windows Classes

Drag and Drop

There has been a number of enhancements or changes to the behavior to the drag and drop functionality for dropping external system files onto window class fields. For example, you can now drop a file into a node in a tree list. The enhancements also bring the platforms more into line.

In the past, regarding drag and drop, when a user wanted to drop files into Omnis they would set kAcceptFiles on a field, and specifically under Windows they would set kAcceptFileData. This would allow them to drag external system files onto the field. When they released the mouse, Omnis would send an evCanDrop, then if this returned kTrue, Omnis sent an evDrop event.

In the case of kAcceptFileData, Omnis would send the filename and filedata in a row, although this was only supported on Windows. In the case of kAcceptFiles, Omnis would send the filename the file extension, but no path, in a row. Dragging system content deeper into a field was not supported, for example, dragging into a tree with nodes.

The updated drag and drop support:

- On evCanDrop for kAcceptFiles the full filenames/path is sent, not just filename
- On evCanDrop for kAcceptFileData the filenames/path and data is sent on macOS as well as Windows.
- evCanDrop is sent as the mouse moves in, out, or within of fields
- evCanDrop and evDrop now will trigger on fields such as trees as the mouse hovers over nodes that can open
- Supporting kAcceptFiles and kAcceptFileData on a field will result in just kAcceptFileData (macOS brought into line with Windows)

Omnis now attempts to read the system dragged filelist & file data on the first entry of a field and disposes these when the drag and drop is complete, or cancelled.

You can disable the new drag and drop support, and revert to the previous evCanDrop support by setting a new flag, classicwindowssystemdragdrop, in the Omnis configuration file (config.json).

Themes

Appearance Theme

Selected Tab Text Color

A new theme color, "colortabselectedtextmacos", has been added to the $apperanace property (and stored in the appearance.json file) which controls the color of the text on a selected tab, on macOS only.

What's New in Omnis Studio 8.1.3

Omnis Studio 8.1.3 contains the following minor enhancements, and some fault fixes which are listed in the Readme text file available with the download.

JavaScript Components

Rich Text Editor

Data Format

The JS Rich Text Editor has a new property, $dataformat, which controls the format of the document data stored in $dataname for the control. It can be one of the following constants:

- **kJSRichTextDataFormatJSON**
  The document data will be stored in the $dataname as JSON, as a Quill 'Delta' object. This is the best option for restoring the data later, as it preserves all formatting.
  When setting the data, you can assign JSON (Delta), HTML or plain text: this should be detected and converted as necessary.
- **kJSRichTextDataFormatHTML**
  The document data will be stored in the $dataname as HTML. This may lose some minor formatting. This format is suitable to use if you are going to use the data elsewhere in your code, but not for storing the document data and restoring into the Rich Text Editor.
  When setting the data, you can assign HTML only.
- **kJSRichTextDataFormatPlain**
  The document data will be stored in the $dataname as Plain text. This will lose most formatting.
  When setting the data, you can assign Plain Text only.

The $dataformat property can be changed in your code to populate the $dataname with data of the specified format. Note that if you do this in a server-executed method, the $dataname won't be updated until the client next contacts the server.

Appending Data

The JS Rich Text Editor has two new methods to allow you to add data either before or after the current content in the Rich Text Editor.

- **$appenddata**(cData, bNewLine)
  $prependdata(cData, bNewLine)
  allow you to append or prepend data to the content in $dataname in a Rich Text editor control.

If you pass bNewLine as kTrue, the data will be added on a separate line. These methods can only be executed on the client.

The data format of the passed data depends on the value of $dataformat. If $dataformat is JSON, the data could be sent as plain text, HTML or JSON (a Quill Delta object).

OW3 Worker Objects

FTP Directory List

FTP does not have a standard syntax for the data returned by the LIST command, so in previous versions the ListDirectory action in the OW3 FTP worker object returned the whole directory list in a single column list. The FTP worker now attempts to parse the results of the ListDirectory action, based on some typical syntaxes supported by many servers. This applies when the list directory action (kOW3ftpActionListDirectory) is being used to generate a detailed list, i.e. when vParam is set to false: the detailed list has 8 columns, as follows:

1. The full text returned by the server. This maintains compatibility with previous versions of the OW3 FTP worker, and may contain additional information not extracted by the parser.
2. The file name.
3. Boolean. True if the entry is probably a directory.
4. Boolean. True if the entry is probably a file.
5. File size in bytes.
6. Modification date of the file.
7. Boolean. True if the modification date is in the local time zone of the client. False means the time zone of the modification date is unknown.
8. If not empty, the server id of the file or directory. A character string.

Window Components

HTML Controls

You can add your own HTML controls to a window class using the oBrowser window component: these have to be added to the htmlcontrols folder. There has been a small enhancement to allow the htmlcontrols folder to be located in the Application Support folder on macOS. This will allow you to add your own HTML controls and the Omnis Runtime app to remain code signed.

The obrowser section of config.json now has a new member:

```
"defaultHtmlcontrolsFolderInDataFolder": false
```

If you set this to true, Omnis looks for the htmlcontrols folder in the data folder rather than the program folder (provided that the obrowser htmlcontrolsFolder member is absent or empty).

What's New in Omnis Studio 8.1.2

Omnis Studio 8.1.2 contains the following minor enhancements, and some fault fixes which are listed in the Readme text file available with the download.

JSON Components

Read-only Properties

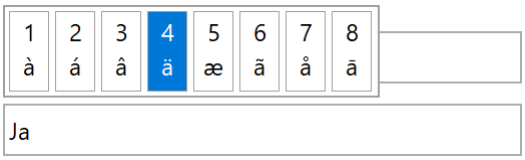
The property object for JSON defined controls has a new Boolean member, **editreadonly,** which can be set to true to make the property read-only. Defaults to false if omitted.

Window Classes

Diacritical Characters

End users can now enter various characters with diacritical marks by using a popup. The new **Diacritical Character** popup is available on end user windows (window classes) in any entry field that accepts text including Entry fields, Multi-Line Entry fields and Combo boxes, as well as in most edit boxes in the Studio IDE, including the Method Editor. (Note *this feature is not available for JavaScript Edit controls in remote forms,* although if your app is running on a mobile device the soft keypad may provide a similar function for entering diacritical characters.)

To enter a diacritical character, the end user needs to hold down the character key, and if the character has additional diacritical options, a popup will be shown containing that character with a range of diacritical marks applied from which a selection can be made. For example, the end user can hold down on the 'a' character key and a small dialog will popup containing a number of diacritical variations for the 'a' character, as shown:



When the popup is shown the end user can press the Left and Right arrow keys to move back and forth, and then press Return to select a character. Clicking the mouse or pointer on a character will also select a character. In addition, the top row in the popup contains a number index and pressing a numeric key will select the corresponding character. The Shift key can also be used along with the character key to enter an uppercase diacritical character.

Pressing Escape or clicking away from the popup will dismiss the popup.

When a character is selected, the popup is dismissed and the last typed character in the original field will be replaced with the selected character.

Popup Content

The popup content varies from language to language. To support different languages a new folder called "Keyboard" has been added to the "Local" folder in the Omnis tree. If you remove this folder no diacritical popup will be shown.

Five languages are supported: English, French, German, Italian, and Spanish. There is a file for each of these languages: en.json, fr.json, de.json, it.json and es.json. When the popup is required, Omnis will load the popup content based on the language of the client, for example, when using English on the client the en.json file is loaded from the Keyboard folder.

You can add more files to this folder to support more languages. The json files have the following structure:

```
"diacritical": {
   "A" : "A À Á Â Ä Æ Ã Å Ā",
   "C" : "C Ç Ć Č",
   "E" : "E È É Ê Ë Ē Ė Ę",
   "I" : "I Î Ï Í Ī Į Ì",
   "L" : "L Ł",
   "N" : "N Ñ Ń",
   "O" : "O Ô Ö Ò Ó Œ Ø Ō Õ",
   "S" : "S Ś Š",
   "U" : "U Û Ü Ù Ú Ū"
}
```

Omnis will search this file for the character key being held down and using the above table present a popup with the various options and index numbers.

You can disable this feature for individual fields in a window class by setting the $disablediacriticalpopup property to kTrue, which is on the Action tab.

macOS Keyboard Layout

On macOS, the user has an option to show a keyboard language menu allowing them to switch between different STANDARD language keyboard input layouts.

With the option **diacriticalpopupuseosxkeyboardlayout** in config.json set to true, depending on the selected keyboard layout in macOS, Omnis will ignore its own 'current' language setting and load a file from the keyboard folder. A 'language to file mapping' must also exists in config.json.

For example, if Omnis is in English but diacriticalpopupuseosxkeyboardlayout is true, and the user has selected French from the standard keyboard layout menu in macOS, Omnis will load the 'fr.json' file for the diacritical character popup content.

```
"diacriticalpopup": {
   "diacriticalpopupuseosxkeyboardlayout": true,
      "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"
   }
```

Diacritical input in the IDE

Assuming the keyboard folder is present in the local folder within the Omnis tree, the IDE will provide the diacritical character popup wherever text entry is required. To disable the feature for the IDE, remove or rename the Keyboard folder.

What's New in Omnis Studio 8.1.1

Omnis Studio 8.1.1 contained a few minor enhancements described below, and some fault fixes listed in the Readme text file available with the download.

OW3 Web Worker objects

An IMAP worker object has been added to the OW3 Worker Objects external package, adding to the existing HTTP, SMTP and FTP worker objects released with Studio 8.1. See the Whatsnew81.pdf or the 'Extending Omnis' online manual for details: http://www.omnis.net/documentation/index.jsp

An example library for each Web or Email protocol has been added to the Samples group under the Hub to demonstrate the use of the new OW3 Worker Objects (look for HTTP, SMTP, FTP and IMAP in the list).

In addition, the RESTful Weather example app now uses the HTTP worker object from the OW3 external package, which is described in the tech note: TNWS0002 "RESTful Web Services: implementing a Client" available here: http://www.omnis.net/technotes/tnws0002.jsp

JSON Control Editor

When you build a new JSON defined JavaScript control you no longer need to restart Omnis to make it available in the Component Store: there is a new 'Reload' button in the JSON Component Editor that loads any new JSON controls that have been built using the 'Build' button.

The 'Reset' button has been renamed 'New' and it loads a new copy of the JSON control template.

SQL Query Builder

Some enhancements have been added to the SQL Query Builder, which is available in the SQL Browser inside the Studio Browser.

A 'Create table class' option has been added to a new 'Other' toolbar menu option for creating a table class from the current query; the option creates a $load method in the table class contains the query from the Query Builder. The option also gives you the option to create a window class and/or a remote form for viewing the data via the new table class; the form contains code which calls the $load method in the table class.

An 'Export Data' option has been added to the 'Other' toolbar menu to allow you to export the results data.

Plus the 'Create Statement on Clipboard' option has been added to the 'Other' menu option; the Omnis code generated by this option is suitable for pasting into an Omnis method.

Cmnd+. keypress on macOS

There is a new setting in the Omnis Configuration file (config.json) in the macOS group. When set to True (the default), the "allowStopInRuntime" option will ensure that the Cmnd+. (Cmnd plus period) key press will stop execution (e.g. break a loop) in a runtime on macOS.

What's New in Omnis Studio 8.1

The following major enhancements were added to Omnis Studio 8.1, as well as several other minor enhancements also listed in this document:

- **Exporting Libraries to JSON**
  you can now export/import Omnis libraries in JSON format which means you can manage and share your Omnis libraries in a third-party VCS repository, such as GIT or SVN
- **Responsive JavaScript Forms**
  JavaScript remote forms now allow you to set custom breakpoints for different screen widths (replacing the existing fixed screen sizes), which means the appropriate form layout and controls will be loaded for the current device; in addition, form controls can transition smoothly when changing the remote form size or orientation
- **JSON Controls**
  you can now define your own remote form controls using JSON or wrap ready-made JavaScript components from a third-party; this provides a new, or alternative method to creating external JavaScript components using C++
- **Headless Omnis Server**
  there is a new "headless" version of the Omnis App Server, available on Linux only, that allows you to deploy your JavaScript Client based web and mobile applications; there is a new Admin tool to help you configure the headless server
- **Code Signed Omnis on macOS**
  the Omnis Studio application package is now code signed on macOS, which provides increased security for you and your end users; consequently, files that may need modification (libraries) are copied to the Application Support folder when Omnis is first run
- **Web and Email Communications**
  there is a new external package, called OW3, that provides a Worker Object containing various methods for performing "low-level" Web- and Email-based communications (HTTP, SMTP, FTP, and IMAP); the new package uses CURL, does not need Java to be installed and configured, and supercedes the previous external commands and web workers
- **Push Notifications**
  Push Notifications are now supported in iOS, Android, and Windows 10 wrappers (version 2.0+) which means you can send messages to clients using your mobile apps; support for notifications is now built into the Wrapper SDKs, and there is a new admin tool under the Tools menu to allow you to set up notifications on clients and the Omnis Server
- **Property Manager** and **Studio Browser**
  the Property Manager has some significant enhancements that will help new and existing users, including a filter for showing a subset or all properties and a Search box for locating specific properties: in addition, the Studio Browser has a Search box to filter the current view to help you locate classes and other items
- **JavaScript Components**
  there are some new properties in JavaScript Edit fields to auto correct, capiltalize, and complete words as the end user types; the automatic correction feature is also available for the editable part of Combo boxes and in Data grids; the JS File control now allows a number of files to be downloaded specified in a list; there is a new

property $showheaderlines for Headed Lists and Tree Lists; if true (the default), header separator lines are drawn in the header

- **Web Services**
  RESTful web services now support POSTs with the content type "application/x-www-form-urlencoded", such as the content type that would be generated by an HTML form on a web page; plus RESTful requests are now queued by the Omnis Server until they succeed
- **Method Editor**
  method lines longer than 255 characters now fully display in the method editor; control characters are now displayed in data or content when inspecting a variable in the Method Editor; inherited methods are no longer prefixed with comments from the inherited method; you can use the shortcut key Ctrl+Shift+I to inherit or override the current method; the Code Assistant now recognises custom properties
- **SQL Workers**
  now support an interim $progress method which can be called whilst the worker is running to provide notifications
- **Window Programming**
  window classes and the majority of window components now have the $alpha property; multi-line fields now have the property $linecount to limit the number of lines of text/data that can be entered into the field; $toplevelhwnd has a new property $screen, that allows you to track the *location* and *dimensions* of the screen, as the window changes position
- **Lists**
  there is a new method $selectduplicates to select duplicate lines in the list; the $first() and $next() list methods now take an additional optional condition parameter which must be met in order to match the first or next line
- **Themes**
  you can now have multiple custom themes, and you can export and import your themes
- **Reports**
  the report class editor toolbar now has Zoom In and Zoom Out buttons which control the DPI value used for report coordinates and rendering fonts; and the A6 paper size has been added
- **Web commands**
  there are two new commands for authentication and executing a HTTP method, and a new parameter UseProxy in HTTPOpen; FTPConnect has a new optional parameter to allow you to specify the Charset
- **Functions**
  there are two new functions to generate 256-bit or 512-bit signatures; and the iso8601 functions provide better handling for hundredths of a second and milliseconds

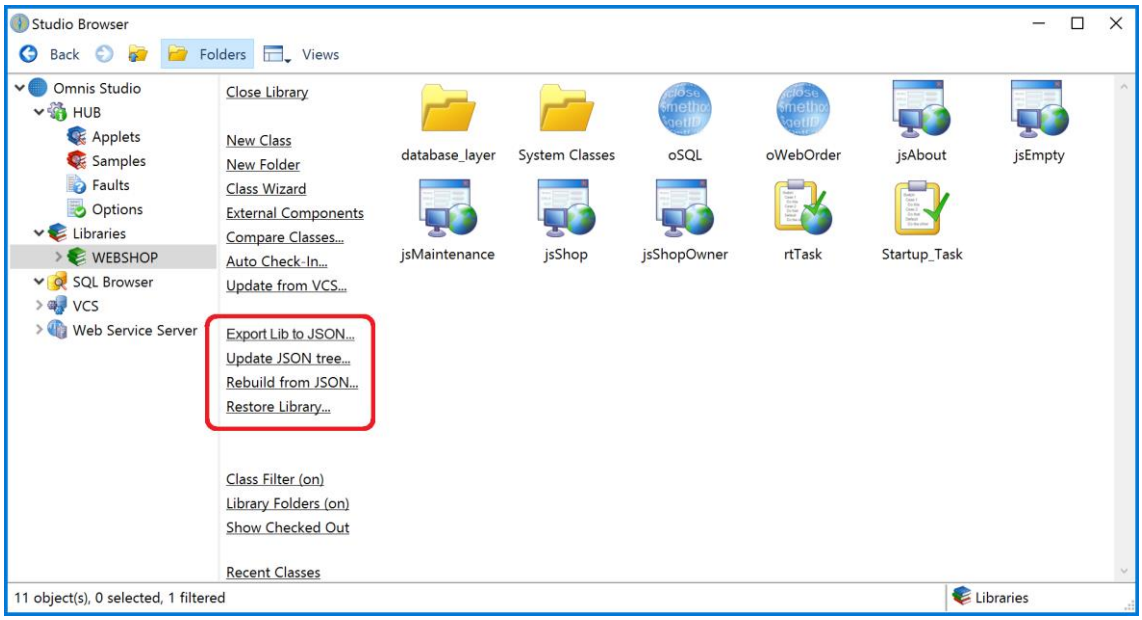
Exporting Libraries to JSON

Omnis Studio 8.1 allows you to *export* an Omnis library to a directory tree containing a number of text files in JSON format representing your library, including all the classes, controls and methods in your library. Additionally, Studio 8.1 will also allow you to *import* an Omnis library from such a JSON tree.

Providing the ability to export and import Omnis libraries in JSON format will allow developers to use a third-party version control system such as GIT or SVN to manage Omnis applications or library source code. In particular, this will allow efficient and secure application development in a team of Omnis developers, as well as the sharing of Omnis libraries and third-party tools among members of the Omnis community.

Exporting Libraries

To export a library to JSON, you need to select the library under the main **Libraries** option in the Studio Browser. After selecting the library the **Export Lib to JSON** option will be visible in the library options, allowing you to export the library to JSON (after you export a library, the Update and Rebuild options will also appear).



If you have multiple libraries open in the Studio Browser, the Export, Update and Rebuild options will apply to the *currently selected library*. By default, different libraries will be exported to different JSON trees, under the export folder, using the library name as the default name for your JSON tree. As you use the Export, Update and Rebuild options, Omnis will maintain an internal table of which library belongs to which JSON export tree to allow you to work on multiple libraries or projects simultaneously.

Export Lib to JSON

The **Export Lib to JSON** option exports the currently selected library to a new JSON tree. The location of the export folder defaults to 'exports' in the main Omnis tree, and the export process automatically creates and names a sub-directory in the export folder using the name of your library.

Update JSON tree

The **Update JSON tree** option exports the library to its associated JSON tree, which in effect will update any classes or methods that have changed, or add any new classes in your library. *You Should Note* that the update option *deletes the existing JSON tree*, and replaces it with a completely new JSON tree built from the updated library.

The update process first checks for any conflicts and reports these if any are found. For example, Omnis will report an error if a JSON file or folder is missing or has been

renamed. You need to rectify these errors before you can update, or you can ignore the conflicts in the error log window and proceed with the update.

Rebuild from JSON

The **Rebuild from JSON** option archives the current library open in the Studio Browser to the 'archives' folder and replaces it with a new library built from the associated JSON tree.

Each time you use the Rebuild option, Omnis places a new copy of the current library in the archives folder and appends a number to the name of the library. The last version of the library in the archive folder is then used during the restore process as the most recent archive.

Once the Rebuild option has been run, the Restore Library option appears.

Restore Library

The **Restore Library** option overwrites the current library in the Studio Browser with the previously archived version.

Library and JSON mapping

The Studio Browser maintains a log of which library maps to which JSON folder, which is essential when working with multiple libraires. A file called 'exports.json' is created in the 'studio' folder that contains the mapping for all your exported libraries, so for each library there is a record of the name and path of the Omnis library file, the name and path of its associated JSON folder, and the path of the archived library, if it exists; note the name of the most recent archive library is used.

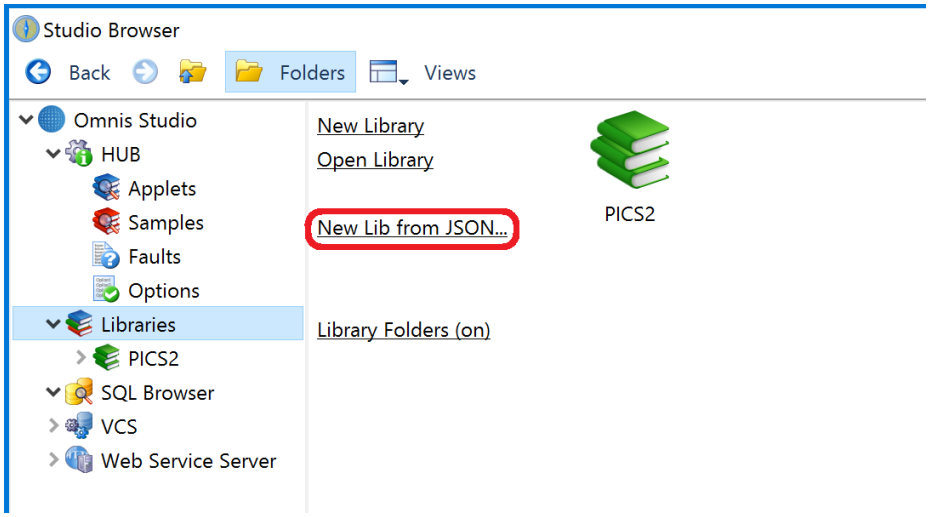
Importing Libraries

You can import a library into the Studio Browser from an existing JSON tree that was previously exported from Omnis Studio using the **Export Lib to JSON** option. For example, you could check out an Omnis JSON tree from a third-party VCS, such as GIT or SVN, and import it into Omnis to start a new project.

Note you cannot open a library from a JSON tree using the standard Open Library option in the Studio Browser (which can only open a .LBS file). You have to import a JSON tree first to create the library before it can be opened in the Studio Browser.

New Library from JSON

To import a library from a JSON tree, you need to select the **Libraries** node in the Studio Browser and click on the **New Lib from JSON** option.



The **New Lib from JSON** option imports a JSON tree that was previously exported from Omnis and creates a new Omnis library file (.LBS). When you have created the new library, its classes will appear in the Studio Browser.

Directory and JSON File Structure

The following sections describe the JSON file & folder structure of a library exported from Omnis Studio using the **Export Lib to JSON** option, which may help you understand how the exported JSON could be managed. Note that all text files exported from Omnis use UTF-8 encoding, including the .json and .omh files, and are formatted suitable for viewing in a text editor.

An Omnis library is represented by a folder that contains the file called 'library.json': this folder has the same name as the library and is referred to as the 'library folder'. library.json contains top-level information about the library, such as the library preferences and version number.

Within the library folder, there is a tree of class directories that represents the folder structure of the Omnis library. Each class in your library has its own directory, and if the class itself is an Omnis folder class, it contains sub-directories for the Omnis classes contained in that Omnis folder.

Each class directory has the same name as the class name (see the note on directory and file naming below). Every class directory contains a JSON file named 'class.json'. This contains top-level information about the class, including:

- Class type
- Class properties
- For classes that support methods: definitions of class and instance variables, and for task and remote task classes, definitions of task variables.

File classes also have a file called 'indexes.json' within the class directory, if the file class defines any indexes.

If the class supports methods, the class directory also contains a JSON file named 'methods.json' provided that there are some class methods. methods.json contains an array of the class methods, where each entry contains various properties of the method and definitions for parameters and local variables.

There is a file in the class directory for each method defined in methods.json, named <method name>.omh (subject to the file naming rules below), that contains the method code. The '.omh' file extension is proprietory to Omnis, but the file format is text like the other files.

If the class can contain objects, then there are two different structures depending on the class type:

- For file, query, schema and search classes, all objects and their properties etc. are in a single file called 'objs.json' in the class directory. objs.json contains an array of objects.
- For all other class types that can have objects, the class directory can have a number of sub-directories:
  - objs
  - bobjs
  - inheritedobjs

The 'objs' directory contains a sub-directory for each object in the class, where the directory name is the object name (subject to the directory naming rules below). Each object sub-directory contains a file named 'object.json' that contains object properties etc, and if the object has methods, there is an identical structure to that used for the class methods: a methods.json file, and <method name>.omh files.

The 'bobjs' directory is only present for window classes (JavaScript forms do not have background objects). It contains a sub-directory for each background object in the class, named using the object ident (subject to the directory naming rules below as older libraries can unfortunately contain objects with duplicate idents). Each background object sub-directory contains a file named object.json that contains object properties, etc.

The ‘inheritedobjs’ directory is only present for classes that support inheritance. It contains a sub-directory for each superclass object that either defines or overrides a method in the subclass. Each sub-directory contains methods.json and <method name>.omh files just like those used for class and object methods, representing the methods defined or overridden for the object.

Binary Data

There are various properties which require a binary representation in the JSON library representation. These are handled in two ways:

1. If Omnis recognises a PNG, e.g. in #ICONS or a report background picture, it outputs a PNG file to the tree, and the JSON contains the name of the PNG file.
2. Otherwise, Omnis outputs the BASE 64 encoding of the binary data to the JSON file.

Directory and File Naming

Where possible, directories and files are named using the Omnis name (class name, object name, object ident, or method name). However, there are some considerations:

1. Although it is not recommended for naming objects in Omnis, class and object names can contain characters that are not allowed in file system names, e.g. path separators for all platforms, ?, *. To cater for this, the JSON library representation escapes these characters as % followed by the 2 lower case hex characters that represent the escaped character. As a consequence, Omnis also escapes the % character.
2. Omnis libraries can contain classes where the names only differ by their case. In addition, they can contain objects with duplicate names. In these cases, the JSON library representation prefixes the name with the string %_<n>_ where <n> is an integer index (for objects this is the order value, and for classes this is a value starting at 1 and incremented for each class with the same case-insensitive name; note that Omnis always exports classes in ascending name order, meaning that the prefix for each class in a set of classes with the same case-insensitive name will be the same each time you export the classes, unless you add or remove a class with the same case-insensitive name).

Library Dependencies

Libraries can depend on other libraries. In many cases, the presence of the external library is not required for Omnis to successfully import or export the JSON library representation. However, there are three cases that affect tokenization, and as a consequence, mean the external library or libraries must be open when exporting or importing a library:

1. Design task. If the design task is in an external library, the external library must be open.
2. Superclass. If the superclass is in an external library, the external library must be open.
3. External file classes. If the code or tokenized properties use a variable in a file class in an external library, the external library must be open.

The export option detects the required external libraries in cases 1-3 above while it generates the JSON library representation. It adds an error to the error list when it encounters a reference to an external library that is not open, and returns kFalse. In addition, if the export succeeds, it adds an array to library.json named “includes”: this is an array of all required external libraries. The import library option will fail if any of the included libraries are not open.

External File classes & Tokenization

By default, Omnis tokenizes variables in external file classes using the file name and a field token. For development, you should use both file and field names (to avoid untokenization issues when the external library is not open), whereas for deployment it might be more desirable for performance to use both file and field tokens.

In Studio 8.0, the only control over these tokenization options is via the browser context menu Retokenize… option. For Studio 8.1, there are some new root preferences that you can use to control this:

- $tokenizeexternalfilenames: If true, Omnis uses tokens rather than text when tokenizing external file names
- $tokenizeexternalfieldnames: If true, Omnis uses tokens rather than text when tokenizing external field names

You can use these preferences when importing a library to control how the output library tokenizes variables in external file classes. The values of these preferences are stored in the "defaults" entry in config.json.

JSON Components

You can now define your own remote form controls using JSON and JavaScript, and use them on JavaScript Remote forms in your web and mobile applications. Using the same technique, you can wrap ready-made JavaScript components available from any third-party, opening up endless possibilities for new controls to use in your Omnis apps.

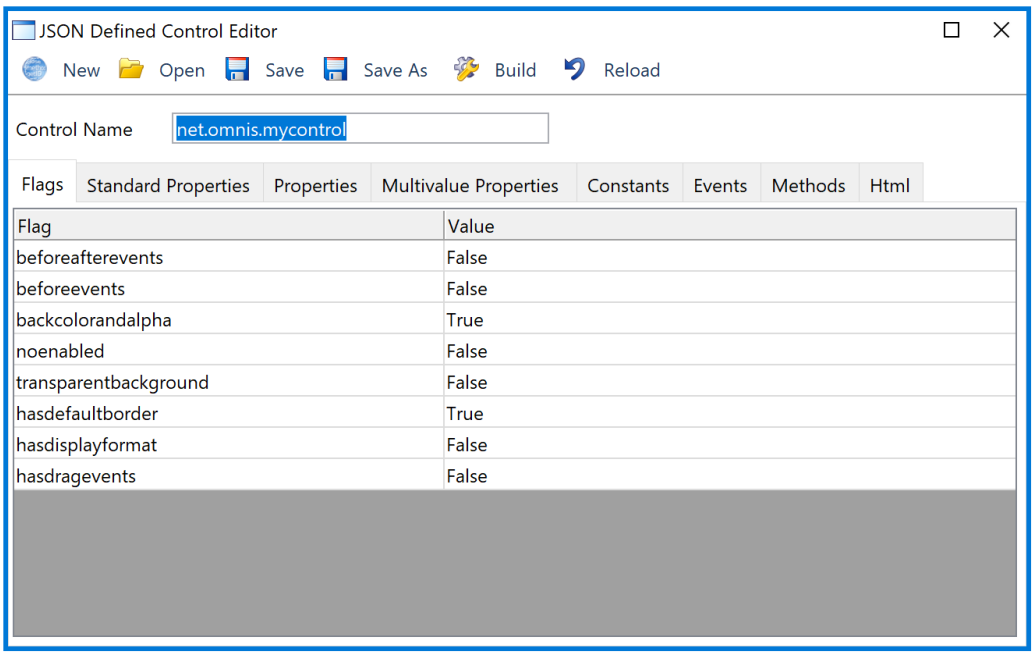
This new method of creating JavaScript components provides an alternative to creating external components using C++ and our JavaScript SDK, which is the current method used for creating JavaScript components. It also means you only need to understand JSON and JavaScript, together with our JavaScript interfaces on the client, in order to create and use the new JSON defined JavaScript controls, either in your own web or mobile apps, or to provide to the wider Omnis community.

Having built a JSON defined component using the JSON Control Editor, the component will appear in the Component Store in a new **JSON Components** group. You can drag the component onto your JavaScript remote form and set its properties using the Property Manager.

The design mode rendering of the JSON controls on a remote form is very basic, and does not reflect the actual control as it might appear on a remote form at runtime, although for some controls that do not require a visual interface this is not a problem. In a later version, we may improve the appearance of the JSON controls in design mode.

JSON Control Editor

A JSON control is defined in a JSON file, called a **JSON Control Definition** (JCD) file, which you can create or edit using any text or JSON editor – if you are very familiar with JSON you may like to create the JCD using an editor. Alternatively, you can create the new JSON controls (create a JCD file) using a new tool available under the Tools>>Add Ons menu, called the **JSON Control Editor.**



The JSON Control Editor contains a template control that has all the necessary properties to create a basic JSON control. The editor allows you to set the properties for the control under each tab. To create a component, you edit the properties, click on Save, click on Build to build the control, and then click on Reload to load the component into the Component Store. The New button removes any changes you have made to the default template and allows you to start again. In order to setup the properties and methods for your control you will need to refer to the later JSON definitions later in this section.

Control Name

The name of the control must be unique, so you will need to change the **Control Name** in the editor (or just accept the default name if you are testing the editor). The default control name is prefixed with 'net.omnis' to show the preferred naming convention, but you should change this to your own company name, e.g. com.mycompany.mycontrol1, or use any appropriate naming convention. If you do use a dot in the control name, Omnis converts it to underscore, since dots cause an issue with the Omnis notation.

Control Properties

The following tabs are available to set the properties of the control:

- **Flags**
  allows you to set whether or not events are enabled, whether or not the control has a transparent background, whether or not drag events are enabled, and so on
- **Standard properties**
  an array of standard properties supported by the control, in addition to the basic properties such as name
- **Properties**
  an object defining the control-specific properties of the control; the name of each member of the properties object is the name of the control property, without the leading $, e.g. id, type, etc.
- **Multivalue properties**
  allows you to set up a control to have multiple values for certain properties
- **Constants**
  an object defining the constants for the control, e.g. value, desc, etc.
- **Events**
  defines the events that the control generates (in addition to those specified by the

flags member) and including the standard events such as evClick; the name includes the "ev" prefix

- **Methods**
  specifies the client-executed methods that the control provides; the method name includes the "$" prefix
- **Html**
  specifies how the initial HTML sent to the client for the control is generated

The Save option places the JSON control file in html/controls folder. The Build option places the JavaScript file for your control in the html/scripts folder in your development tree. It also prompts you to include a reference to the JavaScript file for the control in the **jsctempl.htm** file, which will ensure that the control is available for testing any remote forms that contain the new control.

When you have built a JSON control you need to restart Omnis for it to load. After restarting Omnis, the control will appear under the **JSON Components** tab in the Component Store ready to use in your remote forms. When you deploy your app, you need to place the JSON and JavaScript files in the corresponding folders in your Web Server tree, and check that they are referenced in the html page containing your remote form.

You could open the 'control.json' file created in the JSON Control Editor when you build the control from the template: this file will show you the typical structure of the JSON file required to define a new component.

Using Ready-made JS Components

When using ready-made JS components, that you have obtained from a third-party, you need to add the .js file(s) to the html/scripts folder in the Omnis tree, and any other CSS and image files required for the control need to be put in the appropriate folder(s). You will also need to add any properties, methods, and events in your JS control to the JSON definition file via the JSON Control Editor. There is a tech note on the Omnis website that describes the process of using a ready-made JS component in Omnis:

- TNJC0009: Adding Ready-made JavaScript components to Omnis

You will also need to refer to the JavaScript Control Reference in the JS SDK docs which you can find here:

http://sdkdocs.omnis.net/jssdk

JSON Control Definition

This section describes the different properties that can be defined in the JCD file for the control and edited under the separate tabs in the JSON Control Editor (or when editing separate members using a text editor).

There is a new folder in the Omnis tree, html/controls, which contains a sub-folder for each JSON control you have defined. The names of these sub-folders are not critical, but it makes sense to use the same name as the control name.

The JSON Control Editor will create html/controls folder when you build your first control, otherwise if you are building your own controls you will need to create this folder (note this is not to be confused with the 'htmlcontrols' folder that contains controls that can be loaded in the oBrowser object).

Each control folder must contain a file named control.json. In addition, it can contain PNG files - these can have any name, but they need to comprise a subset of the 16x16, 16x16_2x, 48x48 and 48x48_2x PNG files used for the control icon in the Component Store, and also used when rendering the control on the remote form design window. The PNG files must have the extension .png.

There is a new external component named 'jsControls' in the jscomp folder, which handles all JSON defined controls. It loads and validates the controls at startup. All

controls which pass validation are loaded into the new JSON Components group in the Component Store. If a control fails validation, jsControls opens the trace log, and adds a message to indicate there is a problem with the control. The exact problem can be found in a file called control_errors.txt in the control's folder.

Each control must have a unique name. This is defined in control.json (see below), and you should use a convention similar to Java except that Omnis uses underscore rather than a dot, e.g. net_omnis_control1 could be the name of a control (using dots causes issues in the Omnis notation).

JSON Control Object

Every control has a JSON file called 'control.json' containing a JSON object defining the control. The members of this object are defined in the following sections.

name

The name member is mandatory and it specifies the name of the control; it becomes the external component control name. It is also used to create the JavaScript control class name, as ctrl_<name>.

For example.

```
"name": "net_omnis_control1"
```

In this case the JavaScript control class would be ctrl_net_omnis_control1.

flags

The flags member is mandatory. It is an object that allows certain features of the control to be configured. Each member of flags is optional, and defaults to false if it is omitted. Valid members are:

- **beforeafterevents** and **beforeevents** (are mutually exclusive)
  indicate if the control supports either evAfter and evBefore, or just evBefore respectively. If both are omitted, the control supports neither event (see also the events member)
- **backcolorandalpha**
  indicates if the control has backcolor and backalpha properties.
- **noenabled**
  indicates if the control does not have the enabled property.
- **transparentbackground**
  indicates that the control has a transparent background, and does not have backcolor and backalpha properties. Must not be used with backcolorandalpha set to true.
- **hasdefaultborder**
  indicates if $effect for the control can have the value kJSborderDefault.
- **hasdisplayformat**
  indicates if the control has date and number format properties.
- **hasdragevents**
  indicates if the control has drag events (see also the events member).

For example:

```
"flags": {
    "beforeafterevents": true,
    "backcolorandalpha": true,
    "noenabled": true,
    "hasdefaultborder": false,
    "hasdisplayformat": true,
    "hasdragevents": true
},
```

standardproperties

The standardproperties member is optional. It is an array of standard properties supported by the control; inclusion in the standardproperties member means the control will have the property. These are over and above the basic properties that apply to all controls e.g. active, name, etc.

Valid members of the standardproperties array are: "dataname", "effect", "bordercolor", "borderradius", "linestyle", "font", "textcolor", "align", "fontstyle", "fontsize", "horzscroll", "vertscroll", "autoscroll", "dragmode".

For example:

```
"standardproperties": [
    "dataname",
    "effect",
    "bordercolor",
    "borderradius",
    "linestyle",
],
```

properties

The properties member is mandatory. It is an object defining the control-specific properties of the control. Each member of the properties object is itself an object that contains members that describe the property. The name of each member of the properties object is the name of the control property, without the leading $. Valid members of each property object are:

- **id**
  The identifier of the property. A positive integer. This is mandatory, and it is a critical field in that Omnis stores this value in the copy of the object saved in the class, in order to identify the property. This means you must not change id values after you start to use the control on a remote form. id must be unique for all properties for the control. When jsControls loads the control, it will validate property id uniqueness. It usually makes sense to start numbering your properties at 1.
- **desc**
  The description of the property. A character string. This is mandatory, and is used by the IDE, for example, as the property tooltip in the Property Manager.
- **tab**
  An optional member. A character string that identifies the Property Manager tab to be used for the property. Defaults to the Custom tab if omitted. Otherwise, it must have one of the following values: custom, general, data, appearance, action, prefs, text, pane, sections, java or column.
- **type**
  A mandatory member. A character string that identifies the type of the property. This can be one of the basic types (number, integer, character, boolean or list) or a specific type (color, dataname, font, icon, pattern, fontstyle, linestyle, multiline, set, or remotemenu).
- **runtimeonly**
  An optional member. A boolean which is true to indicate that the property is a runtime only property. Defaults to false if omitted.
- **findandreplace**
  An optional member. A boolean which is true to indicate that the property is searched by find and replace. Defaults to false if omitted.
- **extconstant**
  An optional member. A boolean which is true to indicate that the property is constrained to a range of constants defined by this control. This affects the property manager popup. It can be used for both integer type properties, and set type

properties; in the latter case, the first member of the range must be a constant that has the value zero, and represents the empty set.

- **intconstant**
  An optional member. A boolean which is true to indicate that the property is constrained to a range of constants defined by the Omnis core. This affects the property manager popup. It can be used for both integer type properties, and set type properties; in the latter case, the first member of the range must be a constant that has the value zero, and represents the empty set. extconstant and intconstant must not both be set to true.
- **constrangestart** and **constrangeend**
  These members must be present if either extconstant or intconstant is true. In the case of intconstant, they are integer constant idents that specify the range of constants - you can see the idents for core constants in the $constants group in the notation inspector. In the case of extconstant, these are the names of constants defined by this control; the members of the range are the constants starting with constrangestart, and ending with constrangeend, in the order they occur in control.json. Note that when used with a set, the constant values need to correspond to the bit mask used to represent the set.
- **min** and **max**
  These members are optional, and only apply when the type is integer. They specify minimum and maximum values for the property.
- **initial**
  This member is optional. It can be used to specify an initial value for the property. For number types, it can be a floating point number. For character types, it is a character string. For integer types, it is an integer. For boolean types it is a boolean. The initial value is used, for example, when dragging a new copy of the control out of the Component Store (provided that a copy of the control is not already stored in the Component Store).
- **editreadonly**
  An optional member. A boolean which is true to indicate that the property is a read-only property. Defaults to false if omitted.

For example:

```
"properties": {
    "headercolor": {
        "id": 1,
        "desc": "The header color of the control",
        "type": "color",
        "tab": "appearance",
        "initial": 255
    },
    "headericon": {
        "id": 2,
        "desc": "The header icon of the control",
        "popuptype": "icon",
        "tab": "appearance"
    },
    "rangeofexternalconstants": {
        "id": 3,
        "desc": "Range of external constants",
        "type": "integer",
        "extconstant": true,
        "constrangestart": "kNetOmnisControl1Range1",
        "constrangeend": "kNetOmnisControl1Range3",
    }
```

```
}
```

multivalueproperties

The multivalueproperties member is optional. It allows you to set up a control to have multiple values for certain properties. It is an object with members as follows:

- **itemlistproperty**
  This is mandatory. When a control supports properties with multiple values, the properties are stored in a list. Each row of the list contains the set of properties for a particular tab or column. We call the tab or column (or something else) an item. This property must have type list, and it is automatically hidden from the property manager.
- **itemcountproperty**
  This is mandatory. It is the name of an integer property defined by the properties member, that can be set to specify the number of items in the item list. You can specify a max value for this property in order to restrict the number of items, otherwise it is restricted to no more than 256 items.
- **currentitemproperty**
  This is mandatory. It is the name of an integer property defined by the properties member, that identifies the current item displayed in the property manager, and to which property changes apply to multi-value properties.
- **moveitemproperty**
  This is mandatory. It is the name of an integer property defined by the properties member, that can be used to move the current item to a new position in the item list.
- **properties**
  This is mandatory. It is an object that specifies the properties that have multiple values, and where they are stored in the list. Each member must be the name of a property in the main properties object; the value of each member is the list column in the item list where the property value is stored. It is important not to change the column number once you have started using the control.

For example:

```
"multivalueproperties": {
    "currentitemproperty": "curitem",
    "itemlistproperty": "itemlist",
    "moveitemproperty": "move",
    "itemcountproperty": "itemcount",
    "properties": {
        "mvprop1": 1,
        "mvprop2": 2
        }
    }
}
```

constants

The constants member is mandatory. It is an object defining the constants for the control. Each member of the constants object is itself an object that contains members that describe the constant. The name of each member of the constants object is the name of the constant. Valid members of each constant object are:

- **value**
  The value of the constant. An integer. This is a mandatory member.

- **desc**
  The description of the constant. A character string. This is mandatory, and is used by the IDE for example as the tooltip in the catalog.
- **group**
  The catalog group to which the constant belongs. This is optional. By default, all constants defined for the control belong to the group “RF:jsControls-<control name>”. You can use this member to replace the control name with something else. All constants occurring after the constant with the group specified belong to this group, until a new group is specified (if any).

For example:

```
"constants": {
    "kNetOmnisControlHeaderColor": {
        "value": 123,
        "desc": "The description of this constant"
    },
    "kNetOmnisControl1Range1": {
        "value": 3,
        "desc": "Range constant 1”,
           “group": “Ranges"
    },
    "kNetOmnisControl1Range2": {
        "value": 5,
        "desc": "Range constant 2"
    }
}
```

events

The events member is optional. It specifies the events that the control generates (in addition to those specified by the flags member, i.e. before, after, and drag events). Each member of the events object identifies an event. The name of each member is the name of the event, including the “ev” prefix. Certain standard events can be specified: evClick, evDoubleClick, evTabSelected, evCellChanges, evHeaderClick and evHeadedListDisplayOrderChanged. Valid members of each event object are:

- **id**
  Must not be specified for standard events. Otherwise, this is mandatory, and is the positive integer id of the event. This id must match the event id you use in the JavaScript implementation of the control, and must be unique within the context of this control.
- **desc**
  Must not be specified for standard events. Otherwise, this is mandatory, and is a text string describing the event.
- **parameters**
  The event parameters of the event. This is an array. Each array member is an object with members as follows:
  name
  This member is mandatory. The parameter name. Do not include the p character prefix - Omnis will add this. Note that if you use re-use an event parameter name, then the remaining members of this object are ignored, and overridden by the original definition of the parameter - the first control (or Omnis core) using a name sets the type and description of that parameter.
  type
  This member is mandatory. The data type of the parameter. integer, character, boolean or list.

desc
This member is mandatory. A text string describing the parameter.

For example:

```
"events": {
    "evNetOmnisControlOpened": {
        "id": 1,
        "desc": "The event sent when the control opens",
        "parameters": [
            {
                "name": "name",
                "type": "character",
                "desc": "The name event parameter"
            },
            {
                "name": "name2",
                "type": "integer",
                "desc": "The second event parameter"
            }
        ]
    },
    "evClick": {
        "parameters": [
            {
                "name": "zname1",
                "type": "character",
                "desc": "The zname1 event parameter"
            },
            {
                "name": "zname2",
                "type": "integer",
                "desc": "The zname2 event parameter"
            },
            {
                "name": "horzcell",
                "type": "character",
                "desc": "the horz cell event parameter"
            }
        ]
    }
```

methods

The methods member is optional. It specifies the client-executed methods that the control provides. Each member of the methods object identifies a method. The name of each member is the name of the method, including the "$" prefix. Valid members of each method object are:

- **id**
  This is mandatory, and is the positive integer id of the method. It must be unique within the context of this control. It is used internally by the Omnis core.
- **desc**
  This is mandatory, and is a text string describing the method.
- **type**
  This is mandatory. The return type of the method. integer, boolean, character or list.

- **parameters**
  This member is optional. It is an array describing the parameters of the method. Each member of the array is an object with the following members:
  name
  This is mandatory. The name of the parameter. Omnis will insert a data type character at the start of this name.
  desc
  This is mandatory. A text string describing the parameter.
  type
  This is mandatory.The data type of the parameter. integer, boolean, character or list.
  altered
  Optional. A boolean, default false. If true, the parameter is marked as one that will be altered.
  optional
  Optional. A boolean, default false. If true, the parameter is marked as optional.

For example:

```
"methods": {
    "$mymethod1": {
        "id": 1,
        "desc": "This is my method",
        "type": "integer",
        "parameters": [
            {
                "name": "p1",
                "type": "character",
                "altered": true,
                "desc": "The parameter p1"
            },
            {
                "name": "p2",
                "type": "integer",
                "desc": "The parameter p2",
                "optional":true
            }
        ]
    }
}
```

html

The html member is mandatory. It specifies how the initial HTML sent to the client for the control is generated. It is an object with members as follows:

- **template**
  Mandatory. A character string that is a template for the inner div of the control. For example: <div %o %s data-props='%p' data-mvprops='%m'> :
  jsControls replaces **%o** with the JavaScript client attributes for the client element, which includes the id attribute of the client element: this element must be specified.
  jsControls replaces **%s** with the style attribute for the div, based on the normal Omnis processing and the properties the control supports.
  jsControls replaces **%p** with the control properties that are not multi-value. %p is replaced with a JSON string, representing an object, where each member of the object is named by the property name, with value of the property value. The value may have been mapped by Omnis to what the client will require, for certain property types such as color and icon. The client JavaScript can use this string to create an

object containing its property settings.
jsControls replaces **%m** with the multi-value control properties. %m should be omitted if the control does not use such properties. %m is replaced with a JSON string, similar to %p, except that it is an array of objects, with an array entry for each multi-value item.

- **extrastyles**
  Optional. A string of length up to 255 characters of extra style attributes to include in the style attribute replacing %s in the template, e.g. "margin:2px".
- **padding**
  Optional. An integer used to set padding (in pixels) in the style attribute replacing %s.
- **relativeposition**
  Optional. Boolean, default false. If true, the style attribute replacing %s includes position relative rather than absolute.
- **nowrap**
  Optional. Boolean, default false. If true, the style attribute replacing %s includes white-space nowrap.

For example:

```
"html": {
    "template": "<div %o %s data-props='%p' data-mvprops='%m'></div>",
    "extrastyles":"margin:1px;"
}
```

The resulting inner div for the control looks like this:

```
<div style='position:absolute; top:0px; left:0px; height:106px; width:88px;
   font- family:'Times New Roman',Georgia,Serif; font-size:12pt;font-
   weight:bold; font- style:italic;text-align:right;color:#00FFFF; overflow-
   x:auto; overflow-y:auto;margin:1px;' data-backgroundcolor='#555555;
   rgba(85,85,85,1.0000)' data-dragmode='1' data- effect='1' data-linestyle='1'
   data-bordercolor='16711935' data-
   props='{"headercolor":"#FF0000","headericon":"icons/datafile/omnispic/001663
   n16.png? 46", "rangeofinternalconstants":14, "rangeofexternalconstants":5,
   "headerpattern": 1, "headerfontstyle":4, "headerlinestyle":7,
   "headermultiline":"Lots of text entered like this\rwith multiple\rlines\r",
   "headerset": 13, "headerremotemenu":"NewRemoteMenu", "headerfont":"Courier
   New,Monospace"}' data- mvprops='[{"mvprop1":1,"mvprop2":false,"mvprop3":""},
   {"mvprop1": 2,"mvprop2":true, "mvprop3":"NewRemoteMenu"}, {"mvprop1":
   2,"mvprop2":true,"mvprop3":"aaaa"}]'></div>
```

Note that it is important to use single quotes around the attributes in the template, since JSON includes double quotes. jsControls escapes any single quotes in the JSON it inserts into the place-holders as \u0027.

customtabname

The customtabname member is optional. If specified, it is the name of the custom properties tab for the control shown in the Property Manager.

JavaScript

When you have created a JSON control and added it to your Omnis tree, you can add the supporting JavaScript file to the remote form template in the HTML folder (the JSON Control Editor will do this automatically). To do this you can add:

```
<script type="text/javascript"
   src="scripts/ctl_net_omnis_mycontrol.js"></script>
```

to the scripts section of the jsctempl.htm file (in the html folder) so the control is always included in the test HTML page for your remote form; it also needs to be included in the HTML page serving your deployed web or mobile app.

JavaScript Forms

Responsive Forms

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

Existing remote forms in converted libraries will continue to use the $screensize property to specify the layout for different devices. All new remote forms created in the Studio Browser via the New Class option or the Form wizards are set to the new responsive layout type by default. There is a migration tool, available under the Tools>>Add-Ons menu, that will allow you to migrate existing JavaScript remote forms to the new responsive type. (Note the existing Sync Screen tool only applies to the old $screensize based remote forms.)

Form Layout Type

JavaScript remote forms have a new property, $layouttype, which is a kLayoutType... constant that specifies how the layout of the form is designed. *This property is only assignable when the remote form does not contain any controls* (therefore, you cannot switch an existing remote form in a converted library to the responsive type, if it contains controls). The possible values for $layouttype are:

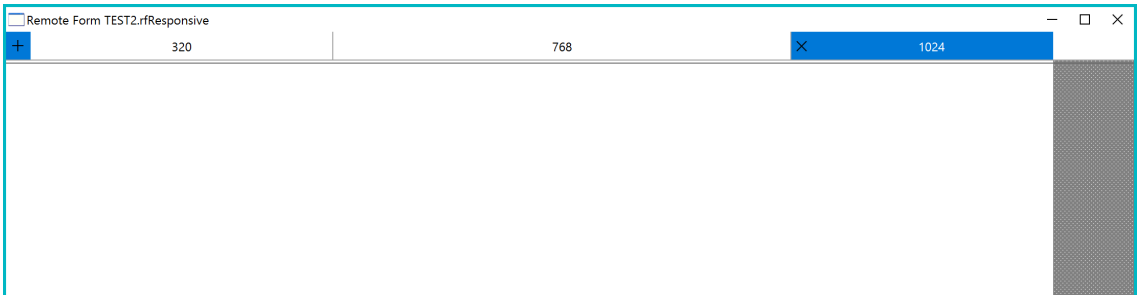
- **kLayoutTypeResponsive**
  The remote form has a responsive layout with layout breakpoints, as specified in the form toolbar and stored in the $layoutbreakpoints property as a comma-separated list. A remote form can have a different layout for the fields and other controls for each breakpoint value.
- **kLayoutTypeScreen**
  This option corresponds to remote forms prior to Studio 8.1, and uses the old $screensize property containing a number of fixed screen sizes. An existing remote form in a library converted to Studio 8.1 will be set to this layout type (you can use the migration tool to convert a form to responsive).
- **kLayoutTypeSingle**
  The remote form has a single layout. This type could be used for applications intended to be deployed on desktop web browsers only: you can use the $edgefloat property for controls to resize or reposition them when the browser window is resized. There is no $screensize property for this layout type and it does not allow breakpoints to be set.

You can return the value of $layouttype in a remote form instance, but you cannot set it in your code. A responsive remote form does not have the following properties, since they are not relevant to responsive design: $resizemode, $screensize, $width, $height, $horzscroll, or $vertscroll.

Creating Responsive Remote Forms

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

A new responsive remote form contains a number of preset layout breakpoints: these are set to **320, 768,** and **1024** which correspond to the relative widths for phones, tablets and desktop computers (the same screen widths available for remote forms in previous versions that used $screensize). You can change these breakpoints to suit the layouts you wish to support in your application (note the default breakpoint values for new remote forms are set in $initiallayoutbreakpoints).



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

Clicking on a layout breakpoint in the toolbar makes it the current layout. You can change, delete or add new layout breakpoints using the toolbar at the top of the remote form design screen, as follows:

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

You can right-click on a breakpoint (which also makes it the current breakpoint) to open a context menu which provides options to edit the breakpoint value and delete the breakpoint (the delete option is enabled only if there are two or more breakpoints).

Deleting Breakpoints

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

Layout Breakpoints

A responsive remote form must have *two or more* layout breakpoints. Layout breakpoints are widths measured in CSS pixels, so they represent logical sizes rather than physical sizes. The JavaScript client chooses the layout for one of the breakpoints

defined in the form based on the logical width of the area in which the remote form is to be displayed in the browser on the device.

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

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

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

Layout Breakpoint Properties

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

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

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

Each layout breakpoint in a remote form has a property $layoutminheight, which is the minimum height of the responsive layout, and shown in design mode as a single horizontal line. When the available client height is larger than this value, controls can float to use the additional vertical space, depending on their $edgefloat properties. The default setting of $layoutminheight is zero which means the minimum height of the form is set to the bottom-most coordinate of all controls plus an additional two pixels for padding. Other non-zero values must be in the range 100 to 32000 inclusive.

What breakpoints should I use?

In general, you need to create a breakpoint for the smallest device within each category of device you wish to support (phones, tablets, desktops). Therefore, the value of the first breakpoint would be the logical width of the smallest phone you wish to support (bearing in mind logical dimensions are not the same as the pixel dimensions, which depend on the density of the screen). For example, the logical dimensions of the iPhone 6 & 7 are 375 x 667, and the Samsung Galaxy S5 & S6 are 360 x 640, so you could set the first layout breakpoint to 360, or perhaps 350 to allow a safe margin and to accommodate form layouts for both phones.

Similarly, to set the layout breakpoint for tablets you should consider the minimum width for the range of tablets you wish to support. The default breakpoints defined in a new remote form (320, 768, and 1024) provide support for a wide range of devices,

both in vertical and horizontal orientations, but you may need to adjust the default breakpoints to suit your requirements.

Adding Controls

You can add JavaScipt controls to a responsive form from the Component Store and set their properties, in exactly the same way as in previous versions. When you add a control to a responsive remote form it is added to the current layout and all other layout breakpoints: initially, a control will be in *the same position in all layouts,* but you can switch to another layout and change its position and other properties for that layout. If you delete a control from one layout it will be removed from all other layouts.

Synchronizing Layouts

You can copy the layout from another layout to the current layout by right-clicking on the background of the remote form, selecting the 'Copy Layout from Breakpoint' option, and choosing the breakpoint value (values other than the current breakpoint are shown). This has the effect of synchronizing the layouts across the different breakpoints, in a similar manner to the Sync Screen tool available for old $screensize based forms.

Control Size and Layout Properties

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

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

Remote Form Inheritance

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

Responsive Form Methods

There are some new notation methods that can be used with a remote form class in order to manipulate its layout breakpoints (note these cannot be used in remote form instances):

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

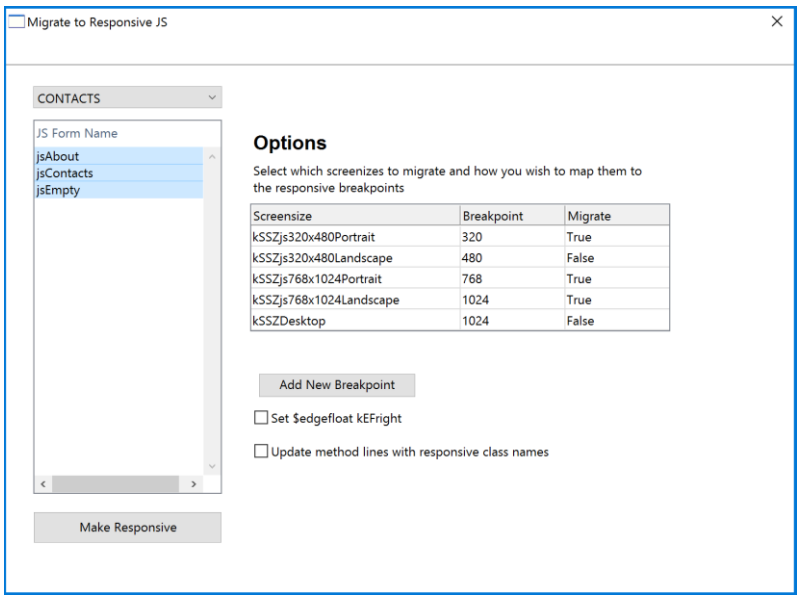
Responsive Form Events

A responsive remote form generates a new event, evLayoutChanged, when the responsive layout breakpoint changes, that is, when a mobile device is rotated, or when a browser window is resized: this event is also triggered when the form opens. This has the event parameter pBreakpoint, which is the integer value of the new layout breakpoint (e.g. 320, 768, or 1024).

In addition, a responsive remote form still generates evScreenOrientationChanged on mobile devices.

Remote Form Migration

There is a migration tool, available under the Tools>>Add-Ons menu, that allows you to convert an existing JavaScript remote form to the responsive form type. The migration tool creates new layout breakpoints corresponding to the old screen sizes available in remote forms in previous versions, and tries to adjust the positioning and layout of fields to fit those breakpoints. The migration tool creates a new responsive remote form with breakpoints and modified screen layouts, based on an existing remote form, and retains the old unmodified form in your library.



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

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

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

The ‘Update method lines with responsive class names’ option will replace all references in your code to the old remote form name to the new name, so your code continues to work.

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

Migration Log and detecting form width

When you have run the migration process, the tool creates a change log which will contain any issues that may need your attention. This may include any places in your code that use the old $screensize constants (kSSZjs...), which no longer apply to responsive forms. In this case, you can use the event evLayoutChanged in the $event method for the form to identify the current breakpoint, for example:

```
On evLayoutChanged
    Calculate iCurrentBP as pBreakpoint
```

The evLayoutChanged event is called automatically when the form first loads as well as when the form resizes. Using the above code you could use the value in iCurrentBP to setup any further sizing or positioning of controls if required.

Component Transitions

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

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

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

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

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

Client Caching

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

The new entry in config.json has the following format:

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

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

Remote Menu Icons

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

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

Subform Sets

There is a new subformset_add constant: **kSFSflagPreventDrag**. If added to the 'flags' parameter of a subformset_add command when using $clientcommand, the user will not be able to drag the windows of the SFS.

Headless Omnis Server

There is a new “headless” version of the Omnis App Server on Linux that allows you to run your JavaScript Client-based web and mobile applications in a headless environment. The headless server is available for Linux only.

A so-called headless Omnis Server installed under Linux does not have a window-based interface, but can be controlled remotely from the command line in a Terminal window on the Linux machine, or you can configure the headless server using a new Admin tool.

Considerations

Console Commands

There is a server config item in config.json ‘headlessAcceptConsoleCommands’, a Boolean. When set to true (the default), the headless server provides a basic command line interface when used in a terminal window.

Functions

The function *isheadless()* returns true when running in the headless server.

sys(231) returns zero in headless server.

sys(233) returns empty in headless server; it returns the title of the main Omnis application window in the full server.

Java

You can start the JVM at startup by setting the ‘startjvm’ in the java section of config.json to true: it cannot be started by any other mechanism on the headless server.

Class Notation

If your Omnis code creates new classes using notation, there is a mechanism to initialise new objects using template files, located in the ‘componenttemplates’ folder in the ‘Studio’ folder. The folders are: componenttemplates/window, componenttemplates/remoteform, componenttemplates/report containing the template files to create window, remote form, and report instances, respectively. Each template file name is complibrary_compcontrol.json, with spaces converted to _ (underscore): it is a copy of an object.json file where only the properties and multivalueproperties members are used. complibrary and compcontrol are the component library and control name.

Restrictions

There are various restrictions or differences from full Omnis Server, as follows:

- Printing images to PDF in the headless server is restricted to PNG images (or true-color shared pictures) only.
- There is no port support.
- You should use the ‘start’ entry in the ‘server’ section of config.json to start the multi-threaded server
- The *Test if running in background* command always sets flag to true in the headless server.
- Several commands and notation methods generate an error if executed in the headless server e.g. open window, $open for a window, etc.

- Picture conversion functions are not supported: pictconvto, pictconvfrom, pictconvtypes, pictformat, pictsize (a runtime error is generated).
- Standard messages generated by the server (OK messages and errors) are sent to the server log file, or could be routed to the Terminal if appropriate

Installing the Headless Server (Linux)

Download the installer from: www.omnis.net/download/

This install assumes you are running as Root or using sudo.

Update your version of Linux using the commands below that correspond to your distribution of linux:

```
Centos/redhat: sudo yum update
Suse: sudo zypper update
Ubuntu/debian: sudo apt-get update
```

Once updated, you will need to install the dependencies that Omnis requires to run, which are as follows:

- Centos/redhat: cups, pango
- Suse/Debian: Runs out of box
- Ubuntu: cups, libpango1.0

Once these are installed you can start the installer:

```
./Omnis-Headless-App-Server-8.1-x64.run
```

Follow through the installer as you would a normal install of Omnis Studio making sure your serial is correct or the install will fail.

For Centos 7 and redhat the service will not automatically start after a reboot, you will need to manually add Omnis (or whatever you called your service) to the service autostart list using the following lines:

```
Sudo /sbin/chkconfig --add homnis
Sudo /sbin/chkconfig --list homnis (This line is to show that you have added
   homnis correctly)
Sudo /sbin/chkconfig homnis on
```

You can now configure the Headless server using the Admin tool, as below.

To summarize the steps for each platform:

CENTOS7 & REDHAT

Required commands for Omnis to run on Centos:

```
Sudo yum update
Sudo yum install cups
Sudo yum install pango
Sudo /sbin/chkconfig --add homnis
Sudo /sbin/chkconfig --list homnis
Sudo /sbin/chkconfig homnis on
```

SUSE

The Headless Server should work out of the box on SUSE, but we would recommend an update just in case:

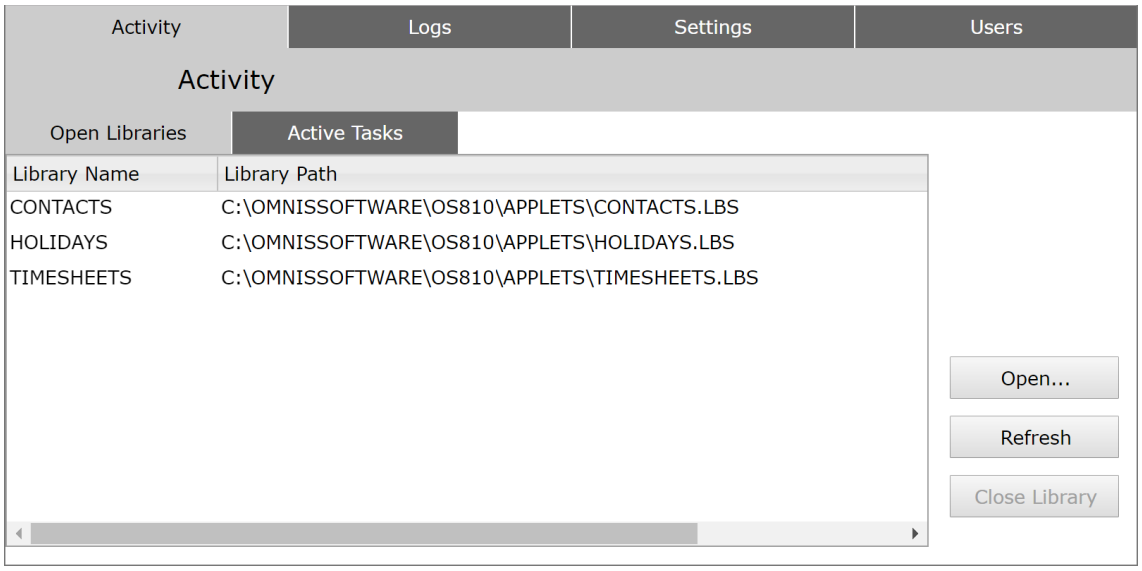
```
sudo zypper update
```

Ubuntu 16.04, 17.04 & DEBIAN 9

```
sudo apt-get update
sudo apt-get install unzip
sudo apt-get install libpango1.0
sudo apt-get install cups
```

Headless Server Admin Tool

There is an Admin tool that you can use to configure the Headless Omnis Server: the Admin tool is implemented as a remote form and can be loaded in a web browser by opening the web page called ‘osadmin.htm’ which is located in the ‘html’ folder of the Omnis Server tree. However, before you can open this page to configure your headless server, you will need to set the “data-webserverurl” parameter in the HTML file to the location of your headless server (URL, IP address or Service name, and Port number, e.g. http://192.1.1.68:5000), and then move the file to a location that allows you to open the file in a web browser and has network access to the headless server (the Headless Omnis Server installer should prompt you to set these options, but you may also like to change them manually).



The Headless Server Admin tool has a number of tabs that let you view or configure the server **Activity, Logs, Settings,** and **Users.** When you first open the admin tool in your browser, you are requested to login: use the default username: **omnis**, password: **0mn1s** (first character is zero). After logging in, you can change the password for the default user, or create other users.

Activity

The **Activity** tab lets you see all **Open Libraries** on the server. You can use the Refresh button to refresh the list.

The **Open** button lets you open a library on the server; note the construct method will be run if present. You can click on a library in the list and close it using the **Close Library** button; note that closing a library will suspend all clients connected to that library.

The **Active Tasks** tab shows all current, active task instances or client connections on the server; you can select a task or connection and view its details. You can kill or close a task instance or connection using the Kill Task button; note that killing a task or connection will suspend the operation of the application for the connected client.

Logs

The **Logs** tab lets you view the logs for the Server:

- **Server**
  provides a log of the headless server activity (the location of the logs can be set under the main Settings tab)
- **Monitor**
  provides a log of all the active client connecttions (task instances)

- **Service**
  provides a log of all the errors or messages generated by the server including any messages in the trace log or information about any web service requests.

Under the Service tab, the **Configure** button lets you set up what messages are recorded in the log, including the attribute "folder" of "logToFile" which is the name of the path relative to the Omnis Server tree where the service logs are generated. These settings are added to the config.json for the server, under the "log" member:

```
"log": {
        "logcomp": "logToFile",
        "datatolog": [
            "restrequestheaders",
            "restrequestcontent",
            "restresponseheaders",
            "restresponsecontent",
            "tracelog",
            "seqnlog",
            "soapfault",
            "soaprequesturi",
            "soaprequest",
            "soapresponse",
            "cors",
            "headlessdebug",
            "headlesserror",
            "headlessmessage",
            "systemevent"
        ],
        "overrideWebServicesLog": true,
        "logToFile": {
            "stdout": true,
            "folder": "logs",
            "rollingcount": 10
        },
        "windowssystemdragdrop": true
    }
```

Settings

Under the **Settings** tab you can specify the location of the Server and Monitor logs, plus the timer period and size of the logs. You can also set up the Server Port, number of Server Stacks, and the Timeslice for the Headless server (and specified in the config.json file), and you can restart the service from here.

The default service name of the Headless server is "homnis" which is specified in the "server" member of the config.json file:

```
{
    "server": {
       …
          "service": "homnis"
```

Users

The **Users** tab lets you update users or create new ones. The default omnis user can be changed here. When checked, the **Re-start Option** will allow a user to restart the server.

Code Signed Omnis (macOS)

The Omnis Studio application package on macOS is now code signed, which provides increased security for you and your end users. A signed application can be trusted to originate from the developer who signed it, and to not have been altered in any way by any third-party, therefore guaranteeing the authenticity of an application. Signed applications within macOS can automatically be granted permissions to perform actions, such as accessing services from the network and running built-in software such as AppleScript commands.

An application can only be signed if its code portion remains unchanged. For the Omnis application, the code portion is located in the Omnis package, e.g.:

```
Omnis\ Studio\ 8.1\ x64.app/Contents/MacOS/
```

Firstruninstall and Application Support folders

Any files that are updated by Omnis must be stored as user application data located in the user's home directory, that is, in the Application Support folder:

```
~/Library/Application Support/Omnis/
```

To do this, when Omnis starts up it will check for the existence of a folder called ‘firstruninstall’ in the macOS folder in the Omnis package. Any items which are contained in this folder are copied by default to a folder in Application Support with the same name as the Omnis package, e.g.:

```
~/Library/Application Support/Omnis/Omnis Studio 8.1 x64
```

The copy will not occur if the destination folder already exists, therefore avoiding any files being overwritten.

This provides a mechanism to place all data folders and their contents into the ‘firstruninstall’ folder, e.g. icons, studio, startup. Once copied into Application Support they are only updated in that location and leave the original macOS folder unchanged and its signature valid.

Updating Components

With the signed version of Omnis Studio, an external or JavaScript component can be added or updated in the user data folder. This allows the signed code part of Omnis to remain unaltered, so it maintains a valid code signature. For example, a standard component can be placed in the following folder:

```
~/Library/Application Support/Omnis/\Omnis\ Studio\ 8.1\ x64/xcomp
```

and a JavaScript component here:

```
~/Library/Application Support/ Omnis/\Omnis\ Studio\ 8.1\ x64/jscomp/
```

If the required folder does not exist it can be created by the user.

The user data folder is always searched first, so if a component with the same name exists in the code section of the Omnis tree the user version will be loaded in preference.

Deployment

When deploying your own application, you can update the distributed files in the Omnis package to include your own libraries and components and to edit the name of the application. Those files placed in the firstruninstall folder will be treated as user data and will be copied to the Application Support folder.

By default, user data for each installation of Omnis goes into a subfolder of Application Support called “Omnis” and the name of the Omnis package is used to provide the folder for the individual installation.

So for example an installation here:

```
/Applications/Omnis Studio 8.1 Beta 3 x64.app
```

Will have a default user data location of:

```
~/Library/Application Support/Omnis/Omnis Studio 8.1 Beta 3 x64
```

To customize the subfolder, edit resource 25599, and to customise the installation folder, edit resource 25600. These resources are located in the Localizable.strings file for the language used, e.g.

```
/Omnis Studio 8.1 x64.app/Contents/Resources/English.lproj/Localizable.strings
```

Both entries are empty for default behavior.

```
"CORE_RES_25599" = "";
"CORE_RES_25600" = "";
```

After you update the Omnis package files, the package will need to be re-signed with your own signing identity. You cannot sign a file that has extended Finder information attributes, so these need to be removed before signing. This can be done recursively over the entire package by using the following command:

```
xattr -r -d com.apple.FinderInfo <package_path>
```

For example:

```
xattr -r -d com.apple.FinderInfo /Applications/My\ Application.app
```

Signing you own application requires a code signing identity which can be generated by adding a development or production certificate via the Certificate section of the Apple developer member center. The machine where signing is to occur must have the certificate and private key installed. To list all valid code signing identities available on a machine, use the following command from the terminal:

```
security find-identity -p codesigning -v
```

Which will, for example, produce the following output with key and identity listed:

```
1) 44FFBA8B7DFFB1AFFF36FD0613D6E5FC61FF8DFF "Certificate"
   (CSSMERR_TP_NOT_TRUSTED)
2) B3EF62FF18E0FFB83D3A8FF3672CF80EFF367FFF "Mac Developer: John Doe
   (24FFEXFF39)"
   2 valid identities found
```

To sign the package use:

```
codesign -f --deep --verbose -s <identity> <package_path>
```

For example:

```
codesign -f --deep --verbose -s "Mac Developer: John Doe (24FFEXFF39)"
   /Applications/My\ Application.app
```

If the command completes with no errors, a similar line to the following should appear in the output:

```
:signed app bundle with Mach-O thin (x86_64) [com.myCompany.MyApplication]
```

The application is now signed and ready for deployment.

Do not subsequently alter the contents of the package as this will invalidate the signature.

You can verify the signature using the following:

```
codesign --display --verbose=4 <package_path>
```

Which will list items such as the signing authority, signing time, etc.

Patching a signed tree

If you wish to distribute an updated Omnis application (the program file), and replace the application in an existing signed Omnis tree, then this can be achieved by doing the following:

- Replace the binary in the original signed tree with the new version.
- Re-sign the Omnis tree with the same signing identity which you used to sign the original tree.
- Take the patched binary out of the tree for distribution.

Components can be patched without re-signing into the xcomp and jscomp folders of the user data location, e.g.:

```
~/Library/Application Support/Omnis/Omnis Studio 8.1 x64
```

Always ensure the tree has a valid signature by running:

```
codesign --display --verbose=4
```

Web and Email Communications

There is a new external package, called **OW3,** that provides various Web Commands that allow you to perform “low-level” Web- and Email-based communications which you can build into your applications. The new OW3 external package in this release contains support for HTTP, SMTP, FTP, and IMAP clients (support for IMAP was added in Studio 8.1.1 patch).

In previous versions of Omnis Studio, the same web and email commands were implemented as external commands, and then as worker objects, available respectively under the *External Commands* group in the Method Editor and the *Web Worker Objects* group in the Object Selection dialog. These implementations will continue to work for backwards compatibility, but the Web Worker Objects are no longer supported in this version, and we therefore recommend you use the new OW3 web and email commands.

By using the OW3 Worker Objects you can execute a long-running task on a background thread, such as running a large mailshot, that reports back to the main thread when the task is complete. In addition, the OW3 new worker objects use the open source *CURL library*, and native secure connection implementations for Windows and macOS, so they should have fewer deployment issues than the implementations available in previous versions.

OW3 Worker Objects

The new web and email commands in OW3 are accessed via a new set of worker objects available under the *OW3 Worker Objects* group in the Object Selection dialog in the Method Editor (not the Web Worker Objects group which contains the existing worker objects). To use the web and email commands, you need to create an Object variable and set its subtype to one of the OW3 worker objects, either the HTTPClientWorker, SMTPClientWorker, or FTPClientWorker object under the OW3 Worker Objects group*.* Having created the variable you can call the web or email commands (methods) using *OBJECTVAR.$methodname.*

The new OW3 worker objects use the same programming model as the SQL DAM workers, and the old OWEB workers, available in previous versions. All OW3 worker objects share the same base functionality, plus they have additional functions specific to their respective web or email protocol.

An example library for each Web or Email protocol has been added to the **Samples** group under the Hub to demonstrate the use of the new OW3 Worker Objects; look for HTTP, SMTP, FTP and IMAP in the list (these were added in Studio 8.1.1).

Base Worker Support

This section describes functionality common to all OW3 worker objects.

Properties

OW3 worker objects all have these properties:

Property	Description
$state	A kWorkerState... constant that indicates the current state of the worker object.
$errorcode	Error code associated with the last action (zero means no error).

$errortext	Error text associated with the last action (empty means no error).
$threadcount	The number of active background threads for all instances of this type of worker object. In this case, type means the type of the subclass of the common base class e.g. HTTP.
$timeout	The timeout (in seconds) for requests. Zero means requests do not time out. The desired value must be set before calling $run or $start. Defaults to 10.
$protocollog	If non-zero, the worker adds a log of protocol activity as a column named log to its wResults row. The desired value must be set before calling $run or $start. Defaults to kOW3logNone. Otherwise, a sum of kOW3log... constants. The kOW3log… constants are described in the Constants section below.
$callprogress	If true, and the worker is invoked to execute asynchronously using $start, the worker periodically generates a notification to $progress as it executes. Must be set before calling $start. The $progress method is described in the Methods section below.
$curloptions	Use this property to set internal CURL options not otherwise exposed by the worker. A two-column list, where column 1 is a number (the CURL easy option number) and column 2 is a string. The internal option must use either an integer or string value. Normally, you would not use this property, but if you do use it, you will need to consult the libcurl header files and documentation to obtain easy option numbers and values. You should use this option with care, as there is a chance you could cause Omnis to crash by passing an incorrect option value.

Constants

Protocol Logging

OW3 worker objects can all use these constants to control protocol logging. Sum the constants to select the desired logging.

Constant	Description
kOW3logNone	No protocol logging occurs. Obviously, this value needs to be used on its own.
kOW3logBasic	Basic protocol information such as headers is logged.
kOW3logData	Application data sent or received is logged up to a maximum of 16k for each direction. If the data is not consistent with UTF-8 encoding, it is logged as a binary dump format rather than character.
kOW3logSecureData	Secure connection data is logged.
kOW3logHTML	The content of the generated log is HTML rather than plain text. This can be written to a file and displayed using OBROWSER on Windows and macOS platforms.

Methods

OW3 worker objects all have the methods described in this section. There are normal methods that you call, and callback methods that you override to receive a notification.

Normal methods

$run

Run the worker on the main thread. Returns true if the worker executed successfully. The callback $completed will be called with the results of the request.

$start

Run the worker on a background thread. Returns true if the worker was successfully started. The callback $completed will be called with the results of the request, or alternatively $cancelled will be called if the request is cancelled.

$cancel

Cancels execution of worker on a background thread. Will not return until the request has been cancelled.

$getsecureoptions

$getsecureoptions([&bVerifyPeer,&bVerifyHost,&cCertFile,&cPrivKeyFile, &cPrivKeyPassword])

Gets the options that affect how secure connections are established.

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

The parameters are:

Parameter	Description
bVerifyPeer	If true, the worker verifies the server certificate. The default is true, and this results in greater security.
bVerifyHost	If true, the worker verifies that the server certificate is for the server it is known as. The default is true, and this results in greater security.
cCertFile	For macOS, the pathname of the .p12 file containing the client certificate and private key, or its keychain name. For other platforms, the pathname of the client certificate .pem file. Empty if a client certificate is not required.
cPrivKeyFile	Ignored on macOS. For other platforms, the pathname of the private key .pem file. Empty if a client certificate is not required
cPrivKeyPassword	The private key password. Empty if a client certificate is not required

$setsecureoptions

$setsecureoptions([bVerifyPeer=kTrue,bVerifyHost=kTrue,cCertFile='',cPrivKeyFile='',cPrivKeyPassword=''])

Sets the options that affect how secure connections are established (call $setsecureoptions before calling $run or $start).

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

The parameters are:

Parameter	Description
bVerifyPeer	If true, the worker verifies the server certificate. The default is true, and this results in greater security.
bVerifyHost	If true, the worker verifies that the server certificate is for the server it is known as. The default is true, and this results in greater security.

cCertFile	For macOS, the pathname of the .p12 file containing the client certificate and private key, or its keychain name. For other platforms, the pathname of the client certificate .pem file. Empty if a client certificate is not required.
cPrivKeyFile	Ignored on macOS. For other platforms, the pathname of the private key .pem file.Empty if a client certificate is not required
cPrivKeyPassword	The private key password. Empty if a client certificate is not required

Callback methods

$completed

When a worker is started using either $run or $start, it reports its completion by calling $completed. Override the $completed method of the worker object to receive this notification. It is called with a single row variable parameter. The columns of the row are specific to each type of worker object, so we describe them in each specific worker object section.

$cancelled

To receive a notification that a request has been cancelled using $cancel, override the $cancelled method of the worker object. It is called with no parameters.

$progress

To receive progress notifications, override the $progress method of the worker object. OW3 worker objects generate notifications to $progress as and when some data has been transferred. Progress notifications will not be generated any more than once a second. Each notification receives a row variable parameter. The row has 4 columns.

Column	Description
downloadTotalBytesExpected	The total number of bytes expected to be downloaded from the server. This may always be zero, for example when the server is using chunked HTTP transfer encoding.
downloadBytesSoFar	The number of bytes downloaded from the server so far.
uploadTotalBytesExpected	The total number of bytes expected to be uploaded to the server.
uploadBytesSoFar	The number of bytes uploaded so far.

HTTP Worker

The HTTPClientWorker provides client HTTP support. For example, you can POST data to a server, execute a RESTful request, or download a file from a server. The following sections describe the HTTP worker properties, constants and methods.

Properties

The HTTPClientWorker has the following properties in addition to the base worker properties described earlier:

Property	Description
$followredirects	If true, the HTTP request will follow a server redirect in order to complete the request. The desired value must be set before calling $run or $start. Defaults to false
$proxyserver	The URI of the proxy server to use for all requests from this object e.g. http://www.myproxy.com:8080. Must be set before

	executing $run or $start. Defaults to empty (no proxy server).
$proxytunnel	If true, and $proxyserver is not empty, requests are tunnelled through the HTTP proxy
$proxyauthtype	The type of HTTP authentication to use when connecting to $proxyserver. A kOW3httpAuthType... constant. kOW3httpAuthType constants are described in the Constants section below.
$proxyauthusername	The user name used to authenticate the user when connecting to $proxyserver using $proxyauthtype.
$proxyauthpassword	The password used to authenticate the user when connecting to $proxyserver using $proxyauthtype.
$responsepath	If not empty, the worker writes response content to the file with this path rather then adding it to the wResults row. The file must not already exist. The desired value must be set before calling $run or $start. Defaults to empty

Constants

The HTTPClientWorker uses the following constants, specified in the iMethod parameter in the $init method, in addition to the base worker constants described earlier:

Constant	Description
kOW3httpMethodDelete	Sends a DELETE method
kOW3httpMethodGet	Sends a GET method
kOW3httpMethodHead	Sends a HEAD method
kOW3httpMethodOptions	Sends a OPTIONS method
kOW3httpMethodPatch	Sends a PATCH method
kOW3httpMethodPost	Sends a POST method
kOW3httpMethodPut	Sends a PUT method
kOW3httpMethodTrace	Sends a TRACE method
kOW3httpAuthTypeNone	Indicates that no HTTP authentication is required (in this case a user name and password do not need to be supplied)
kOW3httpAuthTypeBasic	Indicates that basic HTTP authentication is required
kOW3httpAuthTypeDigest	Indicates that digest HTTP authentication is required
kOW3httpMultiPartFormData	Indicates that HTTP multipart/form-data content is to be sent (this is described below)

Methods

HTTPClientWorker has the methods described in this section in addition to the base worker methods described earlier.

Normal methods

$init

$init(cURI, iMethod, lHeaders, vContent [,iAuthType, cUserName, cPassword])

Called to prepare the object to execute a request, before calling $run or $start.

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

The parameters are:

Parameter	Description
cURI	The URI of the resource, optionally including the URI scheme (http or https) e.g. http://www.myserver.com/myresource. If you omit the URI scheme e.g. www.myserver.com/myresource,the URI scheme defaults to http. You can also include query string parameters if desired e.g. http://www.myserver.com/myresource?param1=test&param2=test
iMethod	A kOW3httpMethod... constant that identifies the HTTP method to perform.
lHeaders	A two-column list where each row is an HTTP header to add to the HTTP request Column 1 is the HTTP header name e.g. 'content-type' and column 2 is the HTTP header value e.g. 'application/json'. If you do not supply the header "accept-encoding" the worker automatically decompresses content compressed using gzip or deflate; however, if you supply this header, the worker does not perform automatic decompression.
vContent	kOW3httpMultiPartFormData or a binary, character or row variable containing content to send with the request. kOW3httpMultiPartFormData means send the content built using the $multipart… methods described below. The worker sends binary data as it is. The worker converts character data to UTF-8 and sends the UTF-8. A row must have a single column containing the path of the file containing the content to send. If you do not specify a content-type header in lHeaders, the worker will generate a suitable type if it recognises the file extension when using a row, or when using a character value it will use text/plain;charset=utf-8. Otherwise, it will use application/octet-stream. In addition, the worker will automatically add a content-length header, so there is no need to pass this in lHeaders.
iAuthType	A kOW3httpAuthType... constant that specifies the type of authentication required for this request. If you omit this and the remaining parameters, authentication defaults to kOW3httpAuthTypeNone.
cUserName	The user name to use with authentication types kOW3httpAuthTypeBasic and kOW3httpAuthTypeDigest.
cPassword	The password to use with authentication types kOW3httpAuthTypeBasic and kOW3httpAuthTypeDigest

NOTE: If you call $init when a request is already running on a background thread, the object will cancel the running request, and wait for the request to abort before continuing with $init.

Example

The following code could be used to prepare an HTTPClientworker object using $init, and then $run can be used to execute the HTTP method. The method returns the content of the web page stored in iURI, e.g. ww.omnis.net.

```
; $execute method
Do method checkHttpObject   ;; sets up the HTTP object ref var
Do method setupLogging   ;; sets up logging based on user choice
If len(iTempContent)
```

```
    Do iHttp.$init(
   iURI,iMethodList.iMethod,iHeaderList,iTempContent,iAuthList.iAuthType,iUser,
   iPassword) Returns lOk
Else
    If iSendContentMode=1
        Do iHttp.$init(
   iURI,iMethodList.iMethod,iHeaderList,row(iContentPath),iAuthList.iAuthType,i
   User,iPassword) Returns lOk
    Else If iSendContentMode=2
        Do iHttp.$buildmultipart(iContentPath)
        Do iHttp.$init(
   iURI,iMethodList.iMethod,iHeaderList,kOW3httpMultiPartFormData,iAuthList.iAu
   thType,iUser,iPassword) Returns lOk
    Else If iSendContentMode=0
        Do iHttp.$init(
   iURI,iMethodList.iMethod,iHeaderList,iContent,iAuthList.iAuthType,iUser,iPas
   sword) Returns lOk
    End If
End If
If not(lOk)
    OK message {Error [iHttp.$errorcode]: [iHttp.$errortext]}
    Quit method kFalse
End If
If pRun
    Do iHttp.$run() Returns lOk
Else
    Do iHttp.$start() Returns lOk
    If lOk
        Calculate $cinst.$objs.ScrollBox.$objs.cancel.$enabled as kTrue
        Calculate $cinst.$objs.ScrollBox.$objs.execute.$enabled as kFalse
        Calculate $cinst.$objs.ScrollBox.$objs.executethencancel.$enabled as
   kFalse
    End If
End If
If not(lOk)
    OK message {Error [iHttp.$errorcode]: [iHttp.$errortext]}
    Quit method kFalse
End If
Quit method kTrue
```

$multipartclear

$multipartclear()

Frees any previously generated multipart/form-data content. Note that calling $run or $start with kOW3httpMultiPartFormData results in the multipart/form-data content being automatically freed after use.

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

$multipartaddfield

$multipartaddfield(cName, cFieldData [,lPartHeaders])

Adds a field part to the multipart/form-data content stored in the worker object. To send this content specify kOW3httpMultiPartFormData as the vContent parameter to $init().

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

The parameters are:

Parameter	Description
cName	The name of the multipart/form-data field part.
cFieldData	The value of the multipart/form-data field.
lPartHeaders	A two-column list where each row is a header to add to the part. Column 1 is the header name and column 2 is the header value.

$multipartaddfile

$multipartaddfile(cName, vFileData [,cFileName='', lPartHeaders])

Adds a file part to the multipart/form-data content stored in the worker object. A file part indicates to the server that a file is being uploaded. To send this content specify kOW3httpMultiPartFormData as the vContent parameter to $init().

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

The parameters are:

Parameter	Description
cName	The name of the multipart/form-data file part.
vFileData	A binary, character or row variable containing the file data for the part. The worker sends binary data as it is. The worker converts character data to UTF-8 and sends the UTF-8. A row must have a single column containing the path of the file containing the content to send. If you do not specify a content-type header in lPartHeaders, the worker will generate a suitable type if it recognises the file extension when using a row, or when using a character value it will use text/plain;charset=utf-8. Otherwise, it will use application/octet-stream.
cFileName	The filename of the part. Must be specified if vFileData is binary or character. If vFileData is a row (identifying a file) then this overrides the default filename (the name of the file).
lPartHeaders	A two-column list where each row is a header to add to the part. Column 1 is the header name and column 2 is the header value.

Callback methods

$completed

The standard $completed callback is passed a row variable parameter with the following columns:

Column	Description
errorCode	An integer error code indicating if the request was successful. Zero means success i.e. the HTTP request was issued and a response received - you also need to check the httpStatusCode to know if the HTTP request itself worked.
errorInfo	A text string providing information about the error if any.
httpStatusCode	A standard HTTP status code that indicates the result received from the HTTP server.
httpStatusText	The HTTP status text received from the HTTP server.
responseHeaders	A row containing the headers received in the response from the HTTP server. The header values are stored in columns of the row. The column name is the header name converted to lower case

	with any - characters removed, so for example the Content-Length header would have the column name contentlength. If the client receives multiple headers with the same name, it combines them into a single header with a comma separated list of the received header values. This is consistent with the HTTP specification.
responseContent	If you have not used $responsepath to write the received content directly to a file, this is a binary column containing the content received from the server.
log	If you used $protocollog to generate a log, this column contains the log data, either as character data, or UTF-8 HTML. Otherwise, the log column is empty.

SMTP Worker

The SMTPClientWorker provides client SMTP support, allowing you to use the worker to send emails, including bulk emails via a mailshot. The following sections describe the SMTP worker properties, constants and methods.

Properties

The SMTPClientWorker has the following properties in addition to the base worker properties described earlier:

Property	Description
$requiresecureconnection	If true, and the URI is not a secure URI, the connection starts as a non-secure connection which must be upgraded to a secure connection (using the STARTTLS command). If it cannot be upgraded then the request fails. Defaults to false
$keepconnectionopen	If true, the worker can leave the connection to the server open when it completes its request. Defaults to false. Note that even when this property is set to true, a protocol error may cause the connection to close. Use true if you are likely to send more emails using the same server fairly soon.
$callmailshotprogress	If true, and the worker is sending a mailshot asynchronously via $start, the worker generates notifications to $mailshotprogress as it executes. $callmailshotprogress must be set before calling $start. Defaults to false

Constants

The SMTPClientWorker uses the following constants in addition to the base worker constants described earlier:

Constant	Description
kOW3msgPriorityLowest	The message has the lowest priority
kOW3msgPriorityLow	The message has low priority
kOW3msgPriorityNormal	The message has normal priority
kOW3msgPriorityHigh	The message has high priority
kOW3msgPriorityHighest	The message has the highest priority

Methods

SMTPClientWorker has the methods described in this section in addition to the base worker methods described earlier.

Normal methods

$init

$init(cURI, cUser, cPassword, vFrom, lTo, lCc, lBcc, cSubject, cPriority, lHeaders, vContent [,bMailshot=kFalse])

Called to prepare the object to execute a request, before calling $run or $start.

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

The parameters are:

Parameter	Description
cURI	The URI of the server, optionally including the URI scheme (smtp or smtps), e.g. smtp://test.com. If you omit the URI scheme e.g. smtp.myserver.com the URI scheme defaults to smtp. If the server uses a non-standard port, you can include it in the URI like this example smtp://smtp.myserver.com:2525
cUser	The user name to be used to log on to the SMTP server.
cPassword	The password to be used to log on to the SMTP server
vFrom	The email address of the message sender. Either a character value e.g. user@test.com or a row with 2 columns where column 1 is the email address e.g. user@test.com and column 2 is descriptive text for the sender, typically their name
lTo	A one or two column list where each row identifies a primary recipient of the message. Column 1 contains the email address e.g. user@test.com and column 2 if present contains descriptive text for the recipient, typically their name
lCc	Empty if there are no CC recipients, or a one or two column list where each row identifies a carbon copied recipient of the message. Column 1 contains the email address e.g. user@test.com and column 2 if present contains descriptive text for the recipient, typically their name
lBcc	Empty if there are no BCC recipients, or a single column list where each row contains the email address of a blind carbon copied recipient of the message e.g. user@test.com. Unlike lTo and lCc, lBcc does not allow more than 1 column, as blind carbon copied recipients are not added to the message header and therefore the descriptive text is not required.
cSubject	The subject of the message
iPriority	A kOW3msgPriority... constant that specifies the priority of the message
lHeaders	A two-column list where each row is an additional SMTP header to send with the message. Column 1 is the header name e.g. 'X-OriginalArrivalTime' and column 2 is the header value e.g. '23:02'
vContent	Message content. Either binary raw content (which the worker sends exactly as it is), or a list to be sent as MIME. See the documentation for the MailSplit command to see how a MIME list is structured; however, note that the charset in the worker MIME list is a kUniType... constant rather than a character string.

bMailshot	Allows a mailshot to be sent (default is kFalse). If true, the worker sends a separate copy of the message to each recipient in the lTo list (so that each recipient cannot see the addresses of the others); only lTo is used, and lCc and lBcc must be empty.

NOTE: If you call $init when a request is already running on a background thread, the object will cancel the running request, and wait for the request to abort before continuing with $init.

Example

The $init method can be used to prepare the SMTPClientWorker object to be executed using the $run or $start method. In this case, iSmtp is an object reference variable with its subtype set to an object class, which has its $superclass set to the SMTPClientWorker in the OW3 worker objects group.

```
; $start method
Do method setupLogging    ;; set up logging
Calculate iSmtp.$timeout as iTimeout   ;; set properties via window fields
Calculate iSmtp.$callprogress as iCallProgress
Calculate iSmtp.$keepconnectionopen as iKeepConnectionOpen
Calculate iSmtp.$requiresecureconnection as iRequireSecureConnection
Calculate iSmtp.$callmailshotprogress as iCallMailshotProgress
Do method $splitaddressentry (iFrom,lFromAddress,lFromDescription) Returns lOk
If not(lOk)
    OK message  {From "[iFrom]" is invalid}
    Quit method kFalse
End If
Calculate lOk as $cinst.$makerecipientlist(lToList,iTo)
If not(lOk)
    OK message  {From "[iTo]" is invalid}
    Quit method kFalse
End If
Calculate lOk as $cinst.$makerecipientlist(lCcList,iCc)
If not(lOk)
    OK message  {From "[iCc]" is invalid}
    Quit method kFalse
End If
Calculate lOk as $cinst.$makerecipientlist(lBccList,iBcc)
If not(lOk)
    OK message  {From "[iBcc]" is invalid}
    Quit method kFalse
End If
If iCallMailshotProgress
    Set reference lMailshotProgressItem to
   $clib.$windows.wMailshotProgress.$openmodal("*",kWindowCenterRelative,$cinst
   ,lToList.$linecount,$cinst)
End If
Do iSmtp.$setMailshotProgressInst(lMailshotProgressItem)
If iNoMIME
    If len(lFromDescription)
        Do iSmtp.$init(
   iServerURI,iUser,iPassword,row(lFromAddress,lFromDescription),lToList,lCcLis
   t,lBccList,iSubject,iPriorityList.iPriorityValue,iExtraHeaderList,lBinConten
   t,iMailshot) Returns lOk
    Else
        Do iSmtp.$init(
   iServerURI,iUser,iPassword,lFromAddress,lToList,lCcList,lBccList,iSubject,iP
```

```
   riorityList.iPriorityValue,iExtraHeaderList,lBinContent,iMailshot) Returns
   lOk
    End If
Else
    If len(lFromDescription)
        Do iSmtp.$init(
   iServerURI,iUser,iPassword,row(lFromAddress,lFromDescription),lToList,lCcLis
   t,lBccList,iSubject,iPriorityList.iPriorityValue,iExtraHeaderList,lMIMElist,
   iMailshot) Returns lOk
    Else
        Do
   iSmtp.$init(iServerURI,iUser,iPassword,lFromAddress,lToList,lCcList,lBccList
   ,iSubject,iPriorityList.iPriorityValue,iExtraHeaderList,lMIMElist,iMailshot)
   Returns lOk
    End If
End If
If not(lOk)
    OK message  {$init error [iSmtp.$errorcode]: [iSmtp.$errortext]}
    Quit method kFalse
End If
If pRun
    Do iSmtp.$run() Returns lOk
Else
    Do iSmtp.$start() Returns lOk
End If
If not(lOk)
    OK message  {$run error [iSmtp.$errorcode]: [iSmtp.$errortext]}
    Quit method kFalse
Else If not(pRun)
    Calculate $cinst.$objs.scrollbox.$objs.cancel.$enabled as kTrue
    Calculate $cinst.$objs.scrollbox.$objs.start.$enabled as kFalse
    Calculate $cinst.$objs.scrollbox.$objs.startthencancel.$enabled as kFalse
End If
Quit method kTrue
```

Callback methods

$completed

The standard $completed callback is passed a row variable parameter with the following columns:

Column	Description
errorCode	An integer error code indicating if the request was successful. Zero means success i.e. the message was successfully sent.
errorInfo	A text string providing information about the error if any.
failedRecipients	If the request was a mailshot, then this column is a three-column list, with a row for each recipient to which the message was not successfully sent. The columns of this list are address (the email address of the failed recipient), errorCode and errorInfo (the latter two columns have the same meaning as the equivalent columns in this row).
log	If you used $protocollog to generate a log, this column contains the log data, either as character data, or UTF-8 HTML. Otherwise, the log column is empty.

$mailshotprogress

If the request is a mailshot, and $callmailshotprogress is kTrue, the worker generates a notification to $mailshotprogress each time it sends (or fails to send) the message to a recipient. $mailshotprogress is passed a row variable parameter with the following columns:

Column	Description
address	The email address of the recipient.
sent	Boolean, true if the message was successfully sent to this recipient

FTP Worker

The FTPClientWorker provides client FTP support, allowing you to use the worker to transfer files. The following sections describe the FTP worker properties, constants and methods.

Properties

The FTPClientWorker has the following properties in addition to the base worker properties described earlier:

Property	Description
$requiresecureconnection	If true, and the URI is not a secure URI, the connection starts as a non-secure connection which must be upgraded to a secure connection (using the STARTTLS command). If it cannot be upgraded then the request fails. Defaults to false
$keepconnectionopen	If true, the worker can leave the connection to the server open when it completes its request. Defaults to false. Note that even when this property is set to true, a protocol error may cause the connection to close. Use true if you are likely to use the same server quite soon.
$servercharset	The character set used by the server to encode file names in commands and file lists. Default kUniTypeAuto (meaning UTF-8 if the server supports it or kUniTypeNativeCharacters if not). Otherwise a kUniType... constant for 8-bit character sets.
$responsepath	If not empty, the worker writes response content to the file with this path rather then adding it to the wResults row. The file must not already exist. The desired value must be set before calling $run or $start. Defaults to empty

Constants

The FTPClientWorker uses the following constants in addition to the base worker constants described earlier. These constants are all actions specified in the iAction parameter used with the $init method to indicate the action to perform, so this section should be read in conjunction with the section describing the $init method:

Constant	Description
kOW3ftpActionPutFile	Upload file data to file cServerPath on FTP server. vParam is file data (binary, character or row). Worker converts character to server character set. Row must have one column (path of file containing

	data to upload). Note that all file transfers use FTP binary mode.
kOW3ftpActionAppendFile	Identical to kOW3ftpActionPutFile except the action appends the file data to an existing file on the FTP server, or creates a new file containing the supplied data if the file does not exist on the FTP server.
kOW3ftpActionGetFile	Download file cServerPath from FTP server. Downloaded file data is either written to $responsepath (if not empty) or returned in the wResults row. vParam is not required. Note that all file transfers use FTP binary mode.
kOW3ftpActionDelete	Delete directory or file cServerPath from the FTP server. vParam is Boolean true if cServerPath is a directory, false if it is a file.
kOW3ftpActionCreateDirectory	Create directory cServerPath on the FTP server. vParam is not required.
kOW3ftpActionListDirectory	List the contents of directory cServerPath on the FTP server returned to wResults.resultList. vParam is Boolean true to list file names only (single column list), or false to get a detailed list with 8 columns: see below.
kOW3ftpActionSetPermissions	Set the permissions of file or directory cServerPath on the FTP server. vParam is a character string specifying the new permissions of the file or directory. Note that not all servers support the SITE CHMOD command used by this action.
kOW3ftpActionExecute	If cServerPath is not empty, CWD cServerPath. Then execute FTP control connection commands in vParam. vParam is either a character string or a single column list of commands. E.g. to rename a file, you could use: RNFR oldname.txt RNTO newname.txt as two lines in the command list. wResults.resultList has a row for each command response.

Directory list for kOW3ftpActionListDirectory

FTP does not have a standard syntax for the data returned by the LIST command, so the FTP worker attempts to parse the results of the ListDirectory action, based on some typical syntaxes supported by many servers. The detailed list has 8 columns, as follows:

1. The full text returned by the server. This maintains compatibility with previous versions of the OW3 FTP worker, and may contain additional information not extracted by the parser.
2. The file name.
3. Boolean. True if the entry is probably a directory.
4. Boolean. True if the entry is probably a file.
5. File size in bytes.

6. Modification date of the file.
7. Boolean. True if the modification date is in the local time zone of the client. False means the time zone of the modification date is unknown.
8. If not empty, the server id of the file or directory. A character string.

Methods

FTPClientWorker has the methods described in this section in addition to the base worker methods described earlier.

Normal methods

$init

$init(cURI, cUser, cPassword, iAction, cServerPath, vParam)

Called to prepare the object to execute a request, before calling $run or $start.

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

The parameters are:

Parameter	Description
cURI	The URI of the server, optionally including the URI scheme (ftp or ftps) e.g. ftp://ftp.myserver.com. If you omit the URI scheme e.g. ftp.myserver.com the URI scheme defaults to ftp
cUser	The user name to be used to log on to the FTP server.
cPassword	The password to be used to log on to the FTP server
iAction	A kOW3ftpAction… constant that specifies the action to perform.
cServerPath	A pathname on the FTP server. Paths are relative to the current working directory on the FTP server. The worker only changes directory if you supply a non-empty cServerPath parameter to kOW3ftpActionExecute, so unless you do this, paths are relative to the root. After changing working directory, if you supply cServerPath prefixed with // then the path is relative to the root, e.g. /myfile or myfile is a path relative to the current working directory, whereas //myfile is a path relative to the root.
vParam	A parameter specific to the action. See the constant descriptions for details of vParam for each action.

NOTE: If you call $init when a request is already running on a background thread, the object will cancel the running request, and wait for the request to abort before continuing with $init.

Example

You could create an FTP client window with various fields for FTP host name, username, password, timeout setting, server character set, and a list of FTP commands or actions as they are defined in the $init() method. A button could initiate the FTP command, executing the appropriate action depending on the one chosen by the end user, using the following code:

```
; start() method
; iFtp is an Object reference variable with the FTPClientWorker as Subtype
; iActionList (List) variable assigned to list of actions on the window
```

```
Do method setupLogging
Calculate iFtp.$timeout as iTimeout   ;; fields on the FTP window
Calculate iFtp.$callprogress as iCallProgress
Calculate iFtp.$keepconnectionopen as iKeepConnectionOpen
Calculate iFtp.$requiresecureconnection as iRequireSecureConnection
Calculate iFtp.$servercharset as iServerCharsetList.C2
Calculate iFtp.$responsepath as iResponsePath
If iActionList.C2=kOW3ftpActionPutFile|iActionList.C2=kOW3ftpActionAppendFile
    If iSendContentMode=0
        ReadBinFile (iContentPath,iContent)
        Do iFtp.$init(
   iServerURI,iUser,iPassword,iActionList.C2,iServerPath,iContent) Returns lOk
    Else
        Do iFtp.$init(
   iServerURI,iUser,iPassword,iActionList.C2,iServerPath,row(iContentPath))
   Returns lOk
    End If
Else If iActionList.C2=kOW3ftpActionSetPermissions
    Do iFtp.$init(
   iServerURI,iUser,iPassword,iActionList.C2,iServerPath,iPermissions) Returns
   lOk
Else If iActionList.C2=kOW3ftpActionExecute
    Do iFtp.$init(
   iServerURI,iUser,iPassword,iActionList.C2,iServerPath,iCommandList) Returns
   lOk
Else If iActionList.C2=kOW3ftpActionListDirectory
    Do iFtp.$init(
   iServerURI,iUser,iPassword,iActionList.C2,iServerPath,iNamesOnly) Returns
   lOk
Else If iActionList.C2=kOW3ftpActionDelete
    Do iFtp.$init(
   iServerURI,iUser,iPassword,iActionList.C2,iServerPath,iPathIsDirectory)
   Returns lOk
Else
    Do iFtp.$init(iServerURI,iUser,iPassword,iActionList.C2,iServerPath)
   Returns lOk
End If
; then $run or $start is called
If not(lOk)
    OK message  {$init error [iFtp.$errorcode]: [iFtp.$errortext]}
    Quit method kFalse
End If
If pRun
    Do iFtp.$run() Returns lOk
Else
    Do iFtp.$start() Returns lOk
End If
If not(lOk)
    OK message  {$run error [iFtp.$errorcode]: [iFtp.$errortext]}
    Quit method kFalse
Else If not(pRun)
    Calculate $cinst.$objs.scrollbox.$objs.cancel.$enabled as kTrue
    Calculate $cinst.$objs.scrollbox.$objs.start.$enabled as kFalse
    Calculate $cinst.$objs.scrollbox.$objs.startthencancel.$enabled as kFalse
End If
Quit method kTrue
```

Callback methods

$completed

The standard $completed callback is passed a row variable parameter with the following columns:

Column	Description
errorCode	An integer error code indicating if the request was successful. Zero means success i.e. the message was successfully sent.
errorInfo	A text string providing information about the error if any.
ftpResponseCode	The FTP response code from the last FTP command executed when performing the action. An integer.
fileData	Used for kOW3ftpActionGetFile only. If you have not used $responsepath to write the file data directly to a file, this is a binary column containing the file data received from the server.
resultList	For kOW3ftpActionList: A single character column list, containing the list entries received from the server. For kOW3ftpActionExecute: A 2 column list, containing an entry for each command supplied in vParam that was successfully executed. Command execution stops as soon as a command fails; the status of the failed command becomes the main error information in the row passed to $completed. Each row of the list contains the ftpResponseCode for the command, and the response text that was received from the server.
log	If you used $protocollog to generate a log, this column contains the log data, either as character data, or UTF-8 HTML. Otherwise, the log column is empty.

Example

Following on from the $init example above, you could create code in the $completed method to handle the response from the FTP server returned in the pResults parameter: the code writes the log to an HTML file and displays it in the oBrowser object.

```
; $completed method
Calculate iResponse as pResults
Calculate iErrorCode as pResults.errorCode
Calculate iErrorText as pResults.errorInfo
If iUsingLogBrowser
    Do FileOps.$deletefile(iLogHTMLPath)
    WriteBinFile (iLogHTMLPath,iResponse.log)
    Calculate iLogBrowser.$urlorcontrolname as
  con("file://",replaceall(iLogHTMLPath," ","%20"))
Else
    Calculate iLog as iResponse.log
End If
Calculate iFailedRecipients as iResponse.failedRecipients
Do $cinst.$redraw()
Calculate $cinst.$objs.tabpane.$currenttab as 3
```

IMAP Worker

The IMAPClientWorker provides client IMAP support, allowing you to use the worker to manage emails stored on an IMAP server. The following sections describe the IMAP worker properties, constants and methods. (IMAP was added in Studio 8.1.1.)

Properties

The IMAPClientWorker has the following properties in addition to the base worker properties described earlier:

Property	Description
$requiresecureconnection	If true, and the URI is not a secure URI, the connection starts as a non-secure connection which must be upgraded to a secure connection (using the STARTTLS command). If it cannot be upgraded then the request fails. Defaults to false
$keepconnectionopen	If true, the worker can leave the connection to the server open when it completes its request. Defaults to false. Note that even when this property is set to true, a protocol error may cause the connection to close. Use true if you are likely to use the same server fairly soon.
$splitfetchedmessage	If true, the worker splits the fetched message into headers and a MIME list for any content. Defaults to true. If false, the worker simply returns the raw fetched message data.
$defaultcharset	Used when kOW3imapActionFetchMessage splits the message, and no character set is specified for a MIME text body part. The character set used to convert to character. Default kUniTypeUTF8. A kUniType... constant (not Character/Auto/Binary).
$removemessageid	If true, the worker removes the Message-id header from the message when performing the action kOW3imapActionAppendMessage. Defaults to true. Duplicating message ids may cause the IMAP server to discard messages with duplicate ids, hence this property.

Constants

The IMAPClientWorker uses the following constants in addition to the base worker constants described earlier. These constants are all actions used with the $init method to indicate the action to perform, so this section should be read in conjunction with the section describing the $init method:

Constant	Description
kOW3imapActionListMailboxes	List mailboxes in reference name cMailboxName. vParam1 specifies the names to list. This becomes the "mailbox name with possible wildcards" parameter of the IMAP LIST or LSUB command (see RFC 3501). vParam2 (optional, default false) is Boolean true to list subscribed mailboxes only.
kOW3imapActionListMessages	Selects mailbox cMailboxName and lists the messages it contains. vParam1 is optional - if present, it is a single column list of additional mail header names to retrieve in addition to the standard mailbox list

Constant	Description
	information e.g. the list could have 2 rows, “Subject” and “X-Priority” to retrieve the message subject and priority for each message.
kOW3imapActionFetchMessage	Selects mailbox cMailboxName and fetches the message with UID vParam1. vParam2 (optional, default false) is Boolean true to fetch message headers only.
kOW3imapActionSetMessageFlags	Selects mailbox cMailboxName and sets flags for message with UID vParam1. vParam2 is a row of flags with values kFalse, kTrue or kUnknown (leave flag unchanged): row(answered, deleted, draft, flagged, seen)
kOW3imapActionAppendMessage	Selects mailbox cMailboxName and appends a message to the mailbox. You can either: Supply the entire message as binary data in vParam1 or Supply a 2 character column list in vParam1 (columns are header name and header value) with binary raw content in vParam2 or Supply a 2 character column list in vParam1 (columns are header name and header value) with the content specified by a MIME list in vParam2. See the documentation for the MailSplit command to see how a MIME list is structured; however note that the charset in the worker MIME list is a kUniType... constant rather than a character string.
kOW3imapActionExecute	If cMailboxName is not empty select mailbox cMailboxName. Then execute IMAP commands in vParam1. vParam1 is either a binary value or a single column list of binary values. wResults.resultList has a row for each command response. Each binary value is an IMAP command to execute, e.g. EXAMINE. You can generate binary values using the correct character set required by the IMAP protocol using the $chartoutf7 method of the IMAPClientWorker. The sequence of actions will stop as soon as an error occurs.

Methods

IMAPClientWorker has the methods described in this section in addition to the base worker methods described earlier.

Normal methods

$init

$init(cURI, cUser, cPassword, iAction, cMailboxName, vParam1, vParam2) Called to prepare the object to execute a request, before calling $run or $start.

Returns Boolean true for success, or returns false and sets $errorcode and $errortext if an error occurs.

The parameters are:

Parameter	Description
cURI	The URI of the server, optionally including the URI scheme (imap or imaps), e.g. imap://ftp.myserver.com. If you omit the URI scheme, e.g. imap.myserver.com, the URI scheme defaults to map
cUser	The user name to be used to log on to the FTP server.
cPassword	The password to be used to log on to the FTP server
iAction	A kOW3imapAction… constant that specifies the action to perform.
cMailboxName	The IMAP mailbox name (ignored for kOWEimapActionListMailboxes). If you are using non-ASCII characters in mailbox names, you may need to normalise the name using the Omnis nfd() or nfc() function before passing it to $init().
vParam1	A parameter specific to the action. See the constant descriptions for details of vParam1 for each action.
vParam2	A parameter specific to the action. See the constant descriptions for details of vParam2 for each action.

NOTE: If you call $init when a request is already running on a background thread, the object will cancel the running request, and wait for the request to abort before continuing with $init.

$chartoutf7

$chartoutf7(cChar)

Returns a binary value (containing 7 bit characters) that is the IMAP UTF-7 representation of cChar (note that IMAP uses a special variant of UTF-7, and this method generates that variant).

The parameters are:

Parameter	Description
cChar	A character string to be converted to IMAP UTF-7

$utf7tochar

$utf7tochar(xUtf7[,bAllowCRLF=kTrue])

Converts IMAP UTF-7 xUtf7 to character and returns the result. Optionally allows CRLF sequences in the data and replaces them with CR in the result (note that IMAP uses a special variant of UTF-7, and this method expects that variant in xUtf7).

The parameters are:

Parameter	Description
xUtf7	A binary value containing IMAP UTF-7 to be converted to character.
bAllowCRLF	If true, CRLF sequences are to be expected in the UTF-7 stream - they are replaced with CR.

Callback methods

$completed

The standard $completed callback is passed a row variable parameter with the following columns:

Column	Description
errorCode	An integer error code indicating if the request was successful. Zero means success i.e. the message was successfully sent.

errorInfo	A text string providing information about the error if any.
resultList	This column receives a list, the content of which depends on the action. The action-specific lists returned here are described below (actions kOW3imapActionSetMessageFlags and kOW3imapActionAppendMessage do not return any data in this column).
log	If you used $protocollog to generate a log, this column contains the log data, either as character data, or UTF-8 HTML. Otherwise, the log column is empty.
<action-specific>	Column 5 is present for certain actions, and contains action-specific data. The action-specific data is described below.

kOW3imapActionListMailboxes:

Column	**Description**
resultList	The list of mailboxes that match the criteria pass to $init(). A 7 column list, with columns as follows (see RFC 3501 for more details - the data in these columns is populated using the LIST or LSUB response): **hasChildren:** Boolean true if the mailbox has child mailboxes. **noInferiors:** Boolean true if it is not possible for any child levels of hierarchy to exist under this name; no child levels exist now and none can be created in the future. **noSelect:** Boolean true if it is not possible to use this name as a selectable mailbox. **marked:** Boolean true if the mailbox has been marked "interesting" by the server; the mailbox probably contains messages that have been added since the last time the mailbox was selected. **unMarked:** Boolean true if the mailbox does not contain any additional messages since the last time the mailbox was selected. **separator:** The mailbox hierarchy delimiter. **mailboxName:** The name of the mailbox.
<action-specific>	No action specific column.

kOW3imapActionListMessages:

Column	**Description**
resultList	The list of messages in the mailbox. This is a list with 9 standard columns, followed by a column for each header specified in vParam2 when calling $init() to prepare for this action. The additional header columns are named by removing - characters from the header name, and converting the result to lower case. The 9 standard columns in the message list are: **UID:** The unsigned integer UID of the message **size:** The size of the message in bytes **internalDate:** The internal date of the message. **answered:** The Boolean answered flag for the message. **deleted:** The Boolean deleted flag for the message. **draft:** The Boolean draft flag for the message. **flagged:** The Boolean flagged flag for the message. **recent:** The Boolean recent flag for the message. **seen:** The Boolean seen flag for the message.

<action-specific>	No action specific column.

kOW3imapActionFetchMessage:

Column	Description
resultList	If $splitfetchedmessage is kFalse, this column is not populated. Otherwise, this column is a 2 character column list of mail headers: Column 1 is the header name. Column 2 is the header value.
<action-specific>	If the action fetches the message content as well as the headers, this column receives the content. It is either: **rawData:** A binary column that receives the un-split fetched message data or **mimeList:** A MIME list containing the content. See the documentation for the MailSplit command to see how a MIME list is structured; however note that the charset in the worker MIME list is a kUniType... constant rather than a character string.

kOW3imapActionSetMessageFlags:

Column	Description
resultList	Not populated.
<action-specific>	No action specific column.

kOW3imapActionAppendMessage:

Column	Description
resultList	Not populated.
<action-specific>	If possible, the action extracts the UID of the appended message from the IMAP server response. The UID is returned to this column (named UID) and is non-zero if the UID could be extracted. (Note that not all servers return the UID of an appended message).

kOW3imapActionExecute:

Column	Description
resultList	A single column list of binary values. Each row of the list contains the sequence of responses returned from the server when executing the corresponding command in the list (or single binary value) passed to $init(). You would typically decode this using $utf7tochar, using the option to expect CRLF and replace with CR.
<action-specific>	No action specific column.

Push Notifications

Push Notifications are now supported in the Android, iOS, and Windows 10 JavaScript Wrappers (version 2.0+) which means you can send messages to any clients that have your mobile app installed (even if it is not running). In this respect, the ability to send push notifications provides a powerful and interactive feature that proactively encourages end users to open and use your mobile app.

A notification or message pushed to a client could include an important news item, a message to users about a new entry into the database, or anything else you want your

end users to know about. You can include a payload of data to send with the notification, which will be passed to your Remote Form, allowing you to react to the user clicking on the notification.

Support for notifications is provided via the *Cloud Messaging* or *Push Notification Service* on the respective platform, which must be enabled in your mobile app project when it is built using the latest JavaScript Wrapper SDK. To setup notifications in your app on Android and iOS, you will need to use *Firebase* from Google: on Windows 10 you need to setup the Push Notification Services in the Store Dashboard.

Push Notifications Admin Tool

In order to manage notifications, it is possible to create groups of devices, and send notifications to particular groups, or individual devices. All functionality can be achieved in your Omnis code (using new properties and methods), or using a new admin tool, called **Push Notifications,** under the **Tools>>Add Ons** menu option on the Omnis menubar. Note the tool is an Omnis library located in the Startup folder which must be present for Push Notifications to work in your mobile apps, including your Omnis code, and for the Omnis App Server configuration to be setup.

Client Command and Methods

There is a new client command **enablepushnotifications** to enable and disable push notifications for mobile apps:

```
$clientcommand("enablepushnotifications", row(bEnable))
```

bEnable: A boolean - kTrue to enable push notifications, kFalse to disable.

Returns: (Boolean) Success. Only if executed in a client method.

In addition, there is a new method $pushnotifycommand which you can use to configure Push Notifications.

For further information about setting up Push Notifications in your mobile apps, and using the client command and methods, see the new Push Notifications document on the wrapper download page, available here:

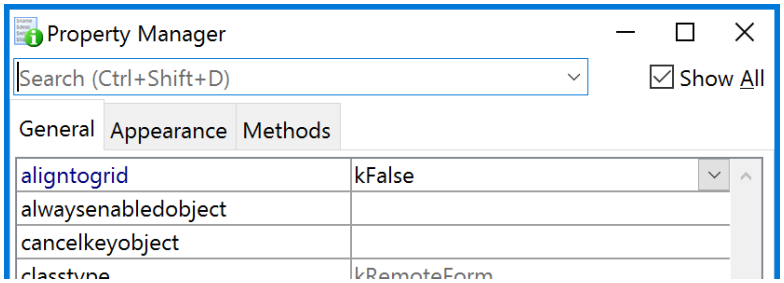
http://www.omnis.net/download/jswrapper.jsp

Property Manager

The **Property Manager** has some significant enhancements that will help new and existing users, including a filter for displaying a “basic” subset or all properties, and a search box for locating specific properties.

Property Filter

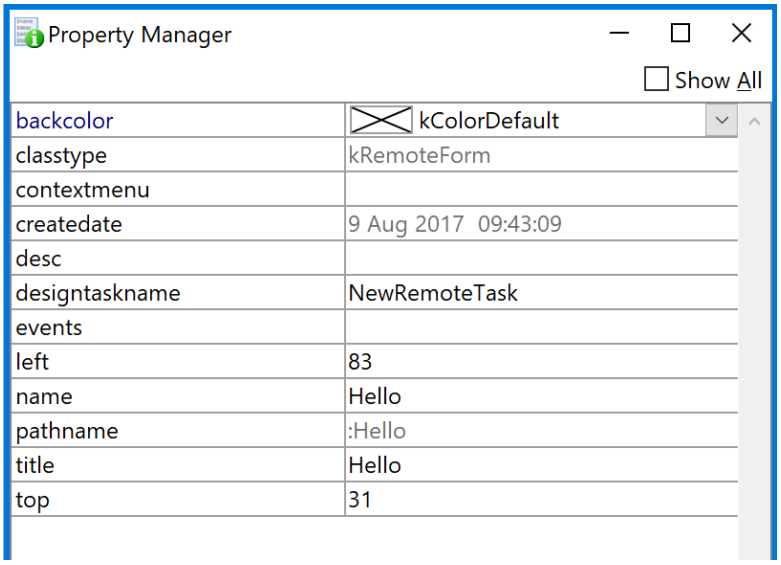
The Property Manager can now display all the properties for an object or a subset of properties. There is a checkbox at the top of the Property Manager window labelled *Show All*, which either shows *all properties* for the current/selected object (listed under specific tabs), or a single filtered list of “basic” properties (tabs are hidden).



The value of the *Show All* filter is saved with the window setup, and its initial value is set according to whether you chose “advanced user” or “new user” in the new

Welcome dialog: advanced sets the "Show All" or unfiltered mode, while new user sets the filter to basic view.

When the Show All option is unchecked, the property list shows a "basic" subset of properties for the current object (selected library, class or form component), or for the current context in the IDE, such as the Omnis preferences. For example, the following image shows the properties for a **remote form** in "basic" mode:



If you use Find & Replace (on the Edit menu) to locate a property, and double-click on the find and replace log to select the property in the Property Manager, the Property Manager automatically switches to "Show All" mode if the property is not part of the basic set.

Modifying the basic set of properties

The basic set of properties is defined in a file called basicproperties.json and stored in the Studio folder under the main Omnis folder. You can modify this file if you want to change the properties shown in the filtered state of the Property Manager. The file is in JSON format, and contains an array of property names which must be lower case, and include the :: prefix if the property name requires one (e.g. some external component properties).

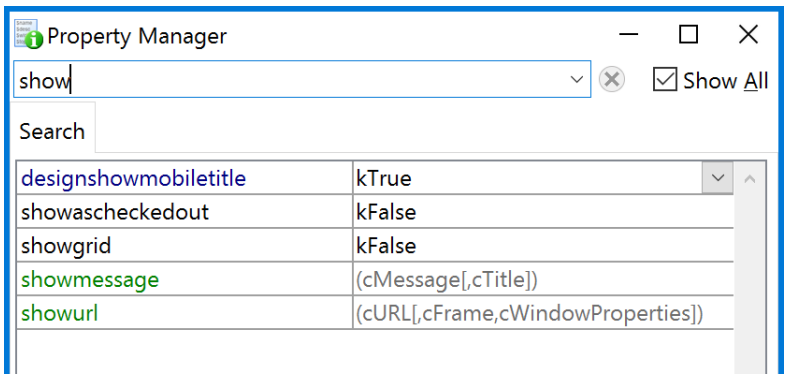
Omnis re-reads this file if it has changed when you uncheck the "all" checkbox in the Property Manager: so checking and unchecking this box forces a re-read. If the file has invalid syntax and cannot be parsed, Omnis writes an error to the trace log, and no basic properties will be displayed.

Property Search

There is a new Search box at the top of the Property Manager window which allows you to search for a property (note the search box is only visible when the "Show All" check box in the Property Manager is checked). You can type a word or part of a word into the search box and the property list will update itself as you type.

The search results are property names that *contain the string* you entered, and they are shown in a single tab named 'Search'. The search results are always sorted by property name, irrespective of the sort list option on the context menu. You can click on a property in the property list and update its value.

For example, entering 'show' into the property search for a remote form will provide a subset of properties containing the word 'show'.



You can use the Backspace to clear a search string character by character, or you can click on the X icon to clear the whole string. The shortcut Ctrl/Cmnd+Shift+D moves the focus to the search box; you can press tab to return the focus to the property list. The 20 most recent searches are saved for re-use, which you can view by clicking on the drop arrow in the search box.

Each keystroke in the Search box performs a search, so there is a delay before a search is saved to the list: the delay defaults to 500ms, but you can change it in the config.json file in the "ide" group: "savePropertySearchDelay".

If you use Find & Replace (on the Edit menu) to locate a property, and double-click on the find and replace log to select the property in the property manager, the property manager clears the search before selecting the property.

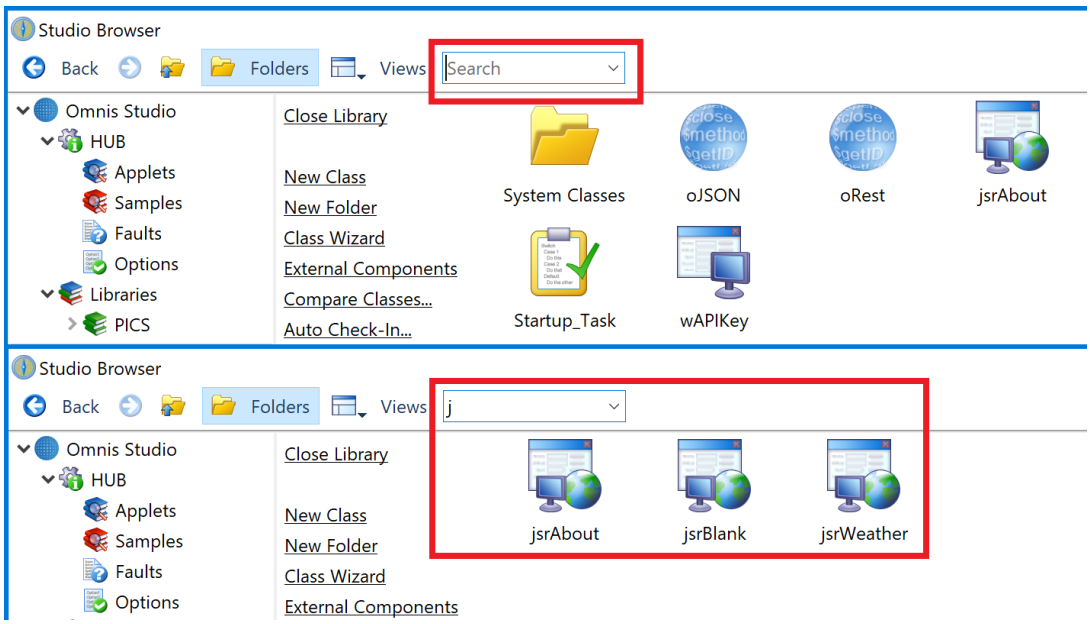
For both the basic mode, and the all mode when search results are being displayed, copy and paste properties are disabled on the context menu.

Studio Browser

Search Filter

The Studio Browser has a new Search box that allows you to filter the objects displayed in the library or class list allowing you to find objects more easily. The new Search filter is available for most of views in the Studio Browser, including Libraries, Classes, SQL sessions, VCS projects, and various parts of the Hub including the Sample apps and Faults.

To search for an item, navigate to the correct view in the Studio Browser, type the first character of the item(s) you are looking for, and the list will instantly redraw, displaying only those items that *start with* the character(s) you typed. For example, in the class list for a library, you could type "j" to find all the classes starting with the letter j.



In most cases the search string you enter is used to find items that start with those characters, except that in the Fault list, under the Hub option, the search string is used to find items that *contain* the search string. The Search box has a dropdown list that stores the last few searches you typed, which you can select from with the pointer.

JavaScript Components

This section describes enhancements made to various JavaScript components, to enhance your web and mobile apps built using remote forms and the JavaScript client.

Edit Controls

There are three new properties for JavaScript Edit controls that allow you to control automatic correction, capitalization and completion of text entered by the end user into an edit field. This functionality is built into the browser whereby text is 'corrected' or updated automatically: note that not all browsers support all of these functions, so you should check support in individual browsers on different platforms. The new properties available for JavaScript Edit controls are **$autocorrect, $autocapitalize,** and **$autocomplete.**

Auto Correction

If true, the $autocorrect property specifies that text entered into a JavaScript Edit control is auto-corrected, *as the end user types into the field*. Currently this feature is only implemented in Safari and iOS browsers (note some browsers, such as Chrome, will highlight incorrectly spelled words, but not correct them automatically). This property defaults to kTrue for backwards compatibility.

Auto Capitalization

The $autocapitalize property controls whether or not text typed into an edit field is capitalized, as appropriate, automatically. Possible values include:

- **kJSAutoCapitalizeSentences**
  Automatically capitalise the first character of new sentences (default and previous behaviour)
- **kJSAutoCapitalizeWords**
  Automatically capitalise the first character of each word
- **kJSAutoCapitalizeNone**
  No automatic capitalization

Auto Completion

The $autocomplete property controls whether or not text is completed automatically. If true, the browser will attempt to complete text based on content previously entered into this type of field (e.g. name type fields will display a list of names based on the first letter typed): the browser will display a list of suggested text for the end user to select from. There may be many possible values for some types of field, which will be based on values previously entered into any website for a field with the same 'autocomplete' value, e.g. 'email'.

Combo boxes and Data grids

Combo boxes and Datagrids have the **$autocorrect** and **$autocapitalize** properties (as described above for Edit controls). For combo boxes, this applies to the edit field section of the control, while for data grids, this applies when editing cells. Consequently, when end users enter text into these controls the text is auto corrected and capitalized if these properties are enabled.

File Control

Multiple File downloads

In previous versions, the JavaScript File control allowed you to download a single file by specifying a row variable containing information about the file to download. You can now use a list, instead of a row, to provide a list of files to be downloaded.

The definition of this list is identical to the row, with the addition of a new 'Identifier' column (the fourth column, of Character type) containing an identifier for each file.

In addition, when multiple downloads occur the UI is no longer blocked by the overlay.

evDownloadSent Event

The File control has a new event, evDownloadSent, which indicates if the download was successful, and identifies the file that was downloaded. The event receives two parameters:

- **pSuccess**
  Whether or not the download was successsfully sent.
- **pIdentifier**
  The value of the Identifier column (the fourth column from the download list, if provided) in the download list/row which was assigned to $dataname to initiate the download which has now been sent.

The evDownloadSent event will be triggered when the server has sent all of the data for a file to the client. Note that Omnis sends the data in a single chunk, so the client may still be processing the data at this point. However, at the point that the event is fired, the server has sent all of the data, and the task variable containing the binary data (or file path) can now be re-used.

Icons Folder Name

You can rename the ‘icons’ folder in Omnis by editing (adding) an entry in the Omnis configuration file (config.json). This may be necessary when you deploy your web or mobile app since Apache often redirects a URL with "/icons/" to the /usr/share/apache2/icons folder, and you would then need to place all the icons for your app in that folder.

There is a new configuration item, you can use in the server group of config.json: "iconsFolder":"omnis_icons" which defaults to "icons" if omitted or the entry is empty.

You are recommended to use the same value for development and runtime, since the folder name is stored in the HTML for each remote form class.

evAfter event queue

When an event is being executed in the JavaScript client, such as a click on a button, a transparent overlay is applied to the whole remote form, to prevent user interaction anywhere else in the form **and** to maintain the Omnis event ordering. If the user clicks on this overlay, the click will be prevented, although most events happen almost instantaneously so in this case the overlay is not displayed. A change has been made to make the effect of clicking on the overlay more intuitive to the user.

For evAfter events that show the overlay, Omnis now shows a feedback effect at the point of the click when the overlay prevents the click, to make it clear to the user that their click was not registered. The feedback effect is a No Entry icon, with "bubble" animation, that appears and disappears directly after the user click.

Previously, if a control with an evAfter had the focus and you then clicked onto another control, if the evAfter took too long to execute, the second control would not receive a click event, as the click will have been captured by the overlay. In this case, the click will now be queued and will fire once the overlay is removed.

Unfortunately, Firefox does not treat the active state of elements in the same way as other browsers. As such, it was not possible to implement these changes for that browser.

Navigation Bar

The Navigation Bar control has a new property, **$pop,** which will "pop" or remove the specified number of items off the navigation stack: it is analogous to clicking on the left or back button, since it allows you to step back in the navigation stack a specified number of times.

The $pop property can only be assigned at runtime. If you try to pop more items off the stack than exist, it will pop everything except the first item. If you assign to $pop, note that the evUserChangedPage event of a linked paged pane will not be triggered.

In addition, you can now set the text or title for the left (back) button for a navbar. If provided, the 6th col in the row assigned to $push allows you to specify the left button text. This can be set to an empty string to default to the previous page's title.

Error Text

There is a new value for the $errortextpos property: **kJSErrorTextPosHidden** which hides the error text, so just the control outline indicates that there is an error (default is a red border). This might be useful where there is limited space to display the error text in the remote form, but you still want to show the end user that there was an error; the style of the error outline is set in the omnis.css style sheet as div.om-error-border.

Text Styles

There is a new font style value, **kLineThrough,** for adding strikethrough to any text. This can be used anywhere there is a property with a font style value, e.g. $fontstyle, or style(kEscStyle,kLineThrough). (This new font style can be applied to any text in JavaScript remote forms, as well as window and report classes.)

Complex Grid

The Complex Grid now has evClick & evDoubleClick events. When clicking on the background of a complex grid row, or a control within the grid which does not have a click event enabled, the evClick or evDoubleClick will be fired. Both of these events receive pLineNumber parameters indicating the line number which was clicked. If no line was clicked (the end user has clicked on empty space), pLineNumber will be 0.

Paged Panes

If a border radius is set on the JavaScript Paged Pane component the rounded corners are no longer drawn in design mode: they are only rendered when the app is run on the client. The rounded corners are not drawn in design mode to allow the full use of the available space within the page pane control while designing the form.

Labels

You can now double-click on a JavaScript form Label to edit its text and corresponding $text property.

Grid Section

Every field or object in a Complex grid has the $gridsection property which tells you the section the object is in (while $gridcolumn which tells you its column). The $gridsection property is now shown in the Notation Inspector so you can inspect its value for an object in a grid section at runtime.

Field List

You can now use the Space bar to select a control in the Field List for a remote form (right-click the form background to open the Field List); the effect of pressing the Space bar will check the box next to the control. Therefore, when the focus is in the Field List, you could use the arrow keys to navigate up and down the list and use Space bar to select a control as required. The Shift-Space keypress allows you to select (or deselect) multiple, discontinuous controls in the list.

Maps

The following is not a new feature but was previously not documented. The JavaScript Map component supports Google Geocoding which allows you to convert a street address into a geographic coordinate like latitude and longitude, which you can use to place a marker on a map, or center the map.

To use the Geocoding function you need to access the Geocoding API which itself requires a separate API Key, in addition to the Maps API key, which you can obtain from Google.

A Search button has been added to the JS Map example available in the Hub to show how you can convert a street address to a latitude:longitude coordinate which can be applied to the $latlong map property. Note the example app places the Geocoding API key in the $userinfo property which is then sent to Google.

Data Grids

Data Grid columns have a new property $columnallownulldateinput to allow a null value to be added to a row of data when the end user tabs out of the last line of the grid to create a new line automatically.

If $columnallownulldateinput is true, and the datatype of the column is Date, cells in the column will default to a value of null when added through the UI. Additionally, if this property is enabled, the end user can change a date to be null by pressing Backspace or Delete while the cell has focus.

If false (the default), the behaviour is unchanged from previous versions. Note is not possible for the end user to input null values into the grid, via the popup date picker, for example.

Web Services

RESTful POSTs

The RESTful API in Omnis now supports the use of POSTs with the content type "application/x-www-form-urlencoded", such as the content type that would be generated by a form on a web page.

Therefore, in addition to URL place-holder parameters, you can now populate parameters using either the query string or application/x-www-form-urlencoded content. You cannot use both the query string and application/x-www-form-urlencoded content. Studio 8.0.2 and earlier just support the query string in addition to URL place-holder parameters.

To use application/x-www-form-urlencoded, set the RESTful input type to application/x-www-form-urlencoded. Omnis then expects application/x-www-form-urlencoded content containing each of the non-optional non-place-holder parameters. The raw application/x-www-form-urlencoded content is also supplied in the pContent parameter of the RESTful method: application/x-www-form-urlencoded content can only be used with HTTP methods that can send content to the server.

Queueing RESTful requests & Licensing

RESTful requests to the Omnis Server consume a web user license for the duration of the request. In previous versions, if all licensed connections were in use when a new RESTful request came into the server, the client received an error. In this version, RESTful requests are now queued internally until they succeed. Note that requests will never be re-queued in a single threaded server (a server where Start server has not been called) since everything executes sequentially.

In addition, there is a new sys function, sys(234), which returns a row of information containing statistics about RESTful requests to the Omnis server. The row has three columns: column 1 is the count of successful calls; column 2 is count of calls resulting in an error; and column 3 is the count of calls internally re-queued because there was not a free user.

RESTful remote task constructor

A RESTful remote task $construct method now receives a row variable parameter with two columns: url and method, where *url* is the partial url starting with the Omnis library component, and *method* is the name of the HTTP method.

Remote Task instances

The $restful and $restfulapiname properties are now available in remote task instances: previously they were only available in remote task classes.

CORS configuration

A template CORS configuration file (cors.json) has been placed in a new folder called 'config' in the 'Studio' folder, containing the required settings to configure CORS for use with Web Services. You can make a copy of this file and place the copy in the Studio folder, making any necessary changes. See the *Extending Omnis* manual for information about setting up and using CORS with Web Services.

Method Editor

Method Lines

Method lines that are longer than 255 characters now fully display in the method editor, right across the code editing area. In previous versions, long method lines were truncated and ended with an ellipsis.

Displaying Control Characters

It is now possible to display control characters in data or content when inspecting a variable in the Method Editor. The Field Value window and Values list window, displayed when you Right-click on a variable, now have a menu that allows you to:

- Show characters normally
- Show all control characters (in this case no line breaking occurs on carriage return for multiline entry fields)
- Show all control characters except carriage return (in this case carriage returns break lines as usual)

The menu also allows you to increase and decrease the font size used for all content except the binary data.

The control characters are displayed using the Unicode page 0x2400 which has visual representations of control characters. In addition to characters 0-0x1f, 0x7f (del) is also treated as a control character.

In addition, the Catalog status bar, Variable value tooltips and Variable context menus always show control characters if present.

The Save Window Setup option for the Values list now saves grid column widths, and the height of the value when using show full value. Save Window Setup for both the Field value window and the Values list window saves the current font size and the option controlling how or if control characters are visible.

Inherited Methods

Comments

The way comments from inherited methods are handled and displayed has changed. In previous versions, when you overrode a method, the new method was prefixed with comments from either the start of the overridden method, or from the overridden external object method. In this version, the new method is no longer prefixed with these comments, instead the comments are shown on the left-hand side of the ‘Notes’ tab in the Variable pane in the Method Editor. Comments for both the immediately overridden method, and for any other implementations further up the inheritance tree are shown.

The new comments field is only present for a method which overrides a superclass method, or for an inherited method. The advantage of this new approach is that the code is not cluttered with comments, and in addition the comments in the new field stay up to date if you edit the comments in a superclass method.

There is a new entry in appearance.json: "overriddenmethodstyle" which is the text style used to indicate an overridden method in the method editor tree. The default is to show methods that override a superclass method in bold.

The method editor now only shows the inherit method option on the method tree context menu when the method does actually override a superclass method.

The method tree context menu now shows the superclass methods… option for methods that override a superclass method. In addition, double click on the method node in the tree opens the superclass methods for both inherited methods (as in previous versions) and overridden methods (new for this change).

You can resize the new field using the mouse. Save window setup saves the width of the new field, provided that the new field is visible when you save the setup.

Inherit or Override method Shortcut

The Inherit method or Override method options are now present in the method editor Modify menu when it is appropriate to include the command. Both have the shortcut Ctrl+Shift+I to inherit or override the current method.

Code Assistant

Custom Properties

The Code Assistant now recognises custom properties, i.e. properties of an instance or an instance object, implemented using two methods, $propname and $propname.$assign.

The Code Assistant combines these into a single property in the list of completions rather than showing the two methods, and provided that the Code Assistant can resolve the parent notation, it will also show $assign and $canassign as possible completions for notation relative to a custom property.

Tabbing Behavior

The behavior of tabbing while using the Code Assistant has changed, although you will need to enable the new behavior in the Omnis configuration file. There is a new item in the 'codeAssistant' entry in config.json to control the behavior when tabbing out of the variable name or calculation box in the method editor.

The new item 'tabAlsoLeavesFieldAfterClosingAssistant' is set to false, by default, but if set to true (and 'oldTabReturnBehaviour' is false) then a tab will close the code assistant and the cursor will move to the next field in the tabbing order.

Renaming Methods

When the focus is on a method name in the method editor tree, pressing Return or Enter allows the selected method name to be edited (provided that it is editable). The line scrolls to view if necessary. If more than one node is selected, nothing happens and Return or Enter does not invoke method name editing. Once the method name has been edited you can press Return or Enter to confirm the edit.

SQL Workers

Additional Notifications

The SQL Worker Objects now support an interim **$progress** method which can be called whilst the worker is running. If implemented in the $callbackinst, the $progress method is called with a row parameter containing a single column, as follows:

- $progress(*row*)
  where *row* is a row parameter containing a single column called 'Progress' which is calculated as a percentage of the total number of SQL queries that will be executed.

Where a work-list/query and bindvar combination is supplied, the total number of queries is calculated by adding the number of times each query will be executed. The received parameter value is suitable for direct assignment to a progress bar component, for example:

```
On evClick
     …
     Do iWorker.$callbackinst.$assign($cinst)
     Do iWorker.$init(lParams) Returns #F
     Do iWorker.$start() Returns #F

; This code appears in the window instance's $progress method
   Do $cwind.$objs.progress.$val.$assign(pRow.Progress)
```

The ‘worker’ sample component supplied with the External Component SDK also demonstrates this functionality.

Window Components

Multi-line Entry Fields

There is a new runtime-only property, $linecount, available for window class edit fields only, that allows you to limit the number of lines of text/data that can be entered into the field. For example, setting $linecount to 2 would only allow 2 lines of text to be entered into the field.

The following example code for the $event method of a multi-line edit field shows how you could prevent users from entering too much text:

```
; set up variable iMaxLines (Integer)
On evClipChangedData,evKey
   Process event and continue
   If $cobj.$linecount>iMaxLines
      Calculate $cobj.$contents as iPrevData
      Sound bell
      Quit event handler
   End If
   Calculate iPrevData as $cobj.$contents
```

Disabling Plug-ins in oBrowser (macOS)

The oBrowser window component has a new property, $disablepluginsmacos, to allow you to disable all plug-ins when running on macOS. This is useful if you want the embedded Safari browser to use the built-in plug-in and not any external plug-in, for example, if you want to display a PDF using the built-in PDF viewer in Safari and not the Adobe PDF viewer installed on the client’s computer.

Note this is not required for Windows since the Chromium Embedded Framework (CEF) in the oBrowser component does not use external plugins installed on a Windows based system.

Headed Lists and Tree Lists

Headed Lists and Tree Lists have a new property **$showheaderlines.** If true (the default), header separator lines are drawn in the header.

Window Programming

Window Transparency

Window classes now have the $alpha property which sets the transparency of the window and all its controls (an integer from 0 to 255, with 0 being completely transparent and 255 opaque). In addition, the majority of the Window class components have the $alpha property which means you can set the transparency of individual window components.

Window instances have the methods $beginanimations() and $commitanimations() which allow you to animate changes to certain properties of some window components including the new $alpha property: other properties supported are $left, $top, $width and $height. For example, you could “fade in” a control by animating a change to its $alpha property, or you could move a component into view by animating a change to its position via its $left and $top properties.

- $beginanimations(iDuration)
  after calling this, assignments to some properties are animated for iDuration milliseconds by $commitanimations()

If you set the same property for an object more than once, the first property change is animated, and then the last property change is animated when the first completes, while property changes between the first and last are ignored. The evAnimationsComplete event (for window instances) is generated after the last property change has completed. This allows you to reverse the effect of an animation (which is the equivalent to the autoreverse/repeat options available on iOS).

Screen Size

There is a new property of $toplevelhwnd, called $screen, that allows you to track the *location* and *dimensions* of the screen, as the window changes position. This could be useful if, for example, a window in Omnis is located on a second monitor and you want to determine its width and height in order to resize or reposition the window.

The value of $left, $top, $width and $height of the screen can be obtained by creating an item reference to this new property, e.g.

```
Set reference toplevelitemref to $cwind.$toplevelhwnd.$ref
Set reference screenref to toplevelitemref.$screen.$ref
```

This is only implemented for macOS and Windows. Other platforms (Linux) will return (0,0,1,1) for (left, top, width, height).

macOS

The screen dimensions on macOS will take account of the menubar and position of the dock and only return the visible screen area.

Windows

Under the Windows OS all Omnis windows are contained within the main Omnis window. Therefore, on Windows the identifier for a window's screen will be the screen containing the main Omnis window.

If the main Omnis window is displayed across multiple screens then the identifier for the screen is that screen which contains the largest area of the main Omnis window. The screen properties will provide the area of the main Omnis window which intersects the visible area of the screen.

List Programming

Select Duplicates

The $selectduplicates method has been added. Its parameters and behavior are exactly the same as $removeduplicates, except the affected lines are selected rather than deleted. The list selection state of non-duplicate lines is cleared.

Note that this can be used in client-executed remote form methods, as well as in server methods.

$first() and $next() Methods

The list methods $first() and $next() now take an additional optional parameter, a *condition* which must be met in order to match the first or next line:

- **$first()**
  LIST.$first([bSelOnly=kFalse, bBackwards=kFalse, condition]) sets $line to first line matching parameters; returns an item reference to the row. If bSelOnly, matches selected lines only; if bBackwards, matches lines in reverse; if condition is present lines must match it
- **$next()**
  LIST.$next(rRow|iRowNumber [,bSelectedOnly=kFalse, bBackwards=kFalse,

condition]) sets $line to the next line after the line identified by the first argument. If iRowNumber is zero, processing starts at $line. See $first for definitions of the other parameters

For example, if name is a column in the list:

```
Set reference item to list.$first(kFalse,kTrue,mid($ref.name,2,1)="M")
Set reference item to list.$next(item,kFalse,kTrue,mid($ref.name,2,1)="M"))
```

Note the new condition parameter can be used in client executed methods in the JavaScript client.

Themes

Custom Themes and Exporting

The Options in the Hub in the Studio Browser are now split into three tabs: Browser, Themes, and Faults.

Under the **Themes** option it is now possible to have multiple custom themes. There is a list of the custom themes currently installed (located in the folder /studio/themes/custom) underneath the themes droplist.

To create a custom theme, press the "Save Current Theme As" button. Once saved, the name you give the theme will then appear in the list of custom themes.

If you are setting a custom theme, you will need to select it first in the list and then press the "Apply Custom Theme" button, since you need to be able to select a custom theme without applying the theme when exporting.

To export custom themes, select the required themes in the list and press the "Export Themes" button. This allows you to select a folder to copy the themes into.

You can also import either a single theme or a folder of themes. Once imported they are copied to the /studio/themes/custom folder and will appear in the list.

Reports

Zoom In/Out

The report class editor toolbar now has Zoom In and Zoom Out buttons which control the DPI value used to convert report coordinates to and from pixels, and the DPI value used to create fonts used in the editor. "Zoom in" increases the DPI value, “Zoom out” decreases it. Note this is for design mode only, and you can zoom through a limited set of DPIs:

- 72 – objects displayed at standard resolution for macOS
- 96 – objects displayed at standard resolution for Windows (with default system scaling of 100%)
- 144 – objects displayed at 2x resolution for macOS
- 192 – objects displayed at 2x resolution for Windows

In addition, if Windows is using a different scaling value, the editor inserts the system DPI into this list at the appropriate point.

These values correspond to the design coordinate system used in Omnis, so on HD displays 96 DPI maps to 192 physical pixels.

You can use Ctrl+ and Ctrl- (Cmnd+/Cmnd- on macOs) to zoom in and zoom out respectively. The current zoom level is saved with the window setup by the save window setup context menu item.

Note that the section bars and the text in the left panel do not increase in height when you zoom. Note also, that zoom does not affect the size of lines drawn in fields on the report - only the text, and in some cases images will scale.

The Modify Report field has a new runtime property, $dpi, that can be assigned to one of the values above.

External Components

If you create external components for reports then you will need to make some changes in order to draw text at the correct DPI. Typically, if the component just displays its name or dataname using the standard interface, you won't need to do anything, as the text DPI will be handled by the Omnis core. Where components that can be placed on reports draw custom text, there are some changes to make in the component:

- There is a new callback ECOgetFontDpi(HWND) that returns the current DPI to use to create fonts - this will return zero unless the component is on a report design window, in which case it will return one of the above values.
- There is a new class GDIfontCreator, that you construct with the HDC for drawing, and the return value from ECOgetFontDpi. This has a method createFont that you then use to create the font rather than calling GDIcreateFont. When you have finished with the font, call GDIdeleteObject as usual. You cannot cache the HFONT generated by createFont in your component.
- If you require font or text metrics, use the HDC versions of GDIfontHeight, GDIfontPart, GDItextWidth etc, with a font created using GDIfontCreator.
- In addition, for more advance use there are classes GDIhdcFontCacheHelper which removes all fonts cached by the Omnis font cache for a particular HDC at the end of the block and GDIoverrideHDCDPI which means that all fonts created for a specific HDC are created at a specified DPI while GDIoverrideHDCDPI is in scope. You need to use GDIoverrideHDCDPI if you are drawing styled text, as styled text drawing may create new fonts. In addition, when drawing styled text, you need to set mFontHdc in the GDIdrawTextStruct, in order for fonts to be created at the correct DPI.
- You can also call GDIcreateDcFont with a DPI parameter to manage fonts yourself.

Paper Size

A6 has been added to the available page sizes. The constant kPaA6 is now available for $paper, an Omnis ($root) preference to set the global page size.

Web Commands

There are two new commands in the Web commands external command set for authentication and executing a HTTP method, and a new parameter, UseProxy, has been added to HTTPOpen. In addition, the FTPConnect command has a new optional parameter to allow you to specify the Charset for the file transfer.

Note these enhancements are in the Web commands located in the External Commands group in the Method Editor, and are not related to the methods in the existing Web Worker objects or new OW3 Worker objects.

HTTPSetAuthentication

HTTPSetAuthentication is a new client command that provides the parameters needed to authenticate an HTTP request with the server; the command only supports HTTP basic authentication, or no authentication. If you use basic authentication, you are recommended to use a secure connection. Use this command to set up authentication after calling HTTPOpen and before calling HTTPMethod. Note that if you do not want to authenticate the request, a new socket created with HTTPOpen defaults to no authentication, so you do not need to call HTTPSetAuthentication in this case.

Syntax

HTTPSetAuthentication (*socket*, *type*, *username*, *password*) Returns *status*

Socket is a long integer field containing the socket number of an open HTTP connection.

Type is a long integer with value zero for no authentication, or 1 for basic authentication.

Username is a character field containing the user name for basic authentication.

Password is a character field containing the password for basic authentication.

Status is an Omnis Long Integer field which receives the value zero for success, or an error code < 0 for failure.

HTTPMethod

HTTPMethod is a new client command that submits to a Web Server an HTTP request to execute a specified HTTP method.

Syntax

HTTPMethod (*socket, uri, method, requesthdrlist, requestcontent, responsestatuscode, responsehdrrow, responsecontent*) Returns *status*

The command requires an existing socket opened with HTTPOpen in order to submit the request. Note that this allows you to sequentially submit more than one request using the same socket connection subject to the rules of HTTP e.g. if the server returns a connection close header in its response, no more requests can be sent on the connection: at this point you need to use HTTPClose to free the socket resources and open a new connection if you want to send more requests to the server. Re-using a connection like this can be a significant performance improvement, especially when using a secure connection, where the connection set-up time is relatively costly.

Socket is a long integer field containing the socket number of an open HTTP connection.

URI is a Character field containing the URI to which the request is addressed. For example, "/default.html", or "/cgi-bin/mycgiscript". Note that if you wish to send query string parameters you must append them to the URI, using the standard ? syntax, e.g. "/default.html?name=test&value=good". You should encode these parameter names and values using CGIEncode.

Method is a Character field containing the HTTP method to be executed. Note that the method is case-sensitive. Standard HTTP methods such as GET and POST need to be specified in upper case.

RequestHdrList is an Omnis list with two character columns. The list contains the HTTP headers to send to the server. HTTPMethod automatically adds a content-length header if you do not specify it in this list, and RequestContent is supplied and not empty.

For example:

Header name	**Value**
User-Agent	My Client
Content-type	text/html

RequestContent is the content to send to the server with the request. It only makes sense to send content with certain methods, e.g. POST. You can supply either character data, which the command converts to UTF-8 before sending, or binary data.

ResponseStatusCode is an integer field into which Omnis returns the HTTP status code of the HTTP request, e.g. 200 for success.

ResponseHdrRow is a row variable which Omnis populates with the headers received in the response from the server. HTTPMethod clears the row and adds columns as

necessary. The column name is the header name converted to lower case, with any - characters removed. In addition, if there are headers with the same name,they are combined into a single header with that name,with comma-separated values.

ResponseContent is a binary or character field into which the command stores the content (if any) received in the response from the server. If this is a character field, the command assumes the content is UTF-8 encoded, and converts from UTF-8 to character before storing the response in the field.

Status is an Omnis Long Integer field which receives the value zero for success, or an error code < 0 for failure.

HTTPOpen

The HTTPOpen command has a new optional Boolean parameter UseProxy.

HTTPOpen (*hostname*[,*service*|*port*,*secure* {Default kFalse},*verify* {Default kTrue}, *useproxy* {Default kTrue}]) Returns *socket*

If UseProxy is kTrue (the default), and proxy server parameters have been set using HTTPSetProxyServer, HTTPOpen connects to the proxy server, setting up a secure tunnel if the proxy server does not have a secure URL, but the requested connection is secure. If kFalse, HTTPOpen connects directly to the specified host and port.

FTPConnect

The FTPConnect web command has a new optional parameter called Charset which specifies the character set used for exchanging pathnames with the FTP server, and for exchanging file character data with the FTP server. Charset can either be kUniTypeAuto, kUniTypeUTF8, kUniTypeNativeCharacters, kUniTypeAnsi..., kUniTypeISO8859_..., or kUniTypeOEM.

The new parameter is added to the end of the existing parameters, therefore the full syntax of the command is now:

- FTPConnect(*ServerAddr,Username,Password*[*,Port,ErrorProtocolText,Secure* {Default zero insecure;1 secure;2 use AUTH TLS}*,Verify* {Default kTrue}*,Charset* {Default kUniTypeAuto}])

If you specify kUniTypeAuto, after FTPConnect establishes a connection, it sends a FEAT command to the server to determine if the server supports UTF8. If the server supports UTF8, then the connection uses UTF8 as the charset, otherwise it uses kUniTypeNativeCharacters.

FTPConnect and TLS

When FTPS is used with the FTP external commands (e.g. FTPConnect to initiate a secure FTP connection), it now resumes the TLS session for data connections. In addition, it automatically sends PBSZ and PROT commands to the server after establishing a secure control connection.

SMTP Workers

Note the following enhancement refers to the existing SMTPClientWorker object in the Web Worker Objects group, not the new OW3 Web Worker object (which also supports mailshots).

Mailshots

The $init method for SMTP worker objects has a new optional parameter, bMailShot, to enable a mail shot to be sent:

- OBJECTVAR.$init(zMessage, cServer [,iSecure=kOWEBsmtpSecureNotSecure, iAuthType=kOWEBsmtpAuthTypeNone, cUser, cPassword, cOAUTH2, cRealm, cNTLMDomain, lProps, bMailShot=kFalse])

If bMailShot is true (the default is false), the worker sends a separate copy of the message to each recipient (so that each recipient cannot see the email address of the other recipients). In this case, only 'to' recipients can be specified.

Functions

SHA functions

There are two new functions to generate almost-unique 256-bit or 512-bit signatures for the supplied binary data.

- **sha256()**
  sha256(*binary*) Returns the 32-byte binary SHA-256 hash of the supplied binary data. Returns #NULL if binary is null or empty.
- **sha512()**
  sha512(*binary*) Returns the 64-byte binary SHA-512 hash of the supplied binary data. Returns #NULL if binary is null or empty.

iso8601 functions

iso8601toomnis() will now parse a date string in the form:
YYYY-MM-DDTHH:MM:SS.FFFZ
rounding FFF into the hundredths field appropriately. The representations in the date string are:

YYYY-MM-DD	Date: Year, month, day
T	Date time delimeter
HH:MM:SS.FFF	Time: Hours, minutes, seconds, fractions of a second
Z	UTC offset: Z means zero UTC offset, or add +/- offset

For more information about ISO8601 look up: https://en.wikipedia.org/wiki/ISO_8601

omnistoiso8601() has a new Boolean parameter that indicates that hundredths are to be output as milliseconds:

- omnistoiso8601(dOmnisDateTime,bNeedTime[,cErrText,bMillis=kFalse])
  Converts dOmnisDateTime (in UTC) to ISO8601 date/date-time (depends on bNeedTime) and returns it. Returns NULL and cErrText on error. Rounds down hundredths if bMillis is false.

Note that bMillis only applies when bNeedTime is kTrue. If bMillis and bNeedTime are both kTrue, then the function always outputs milliseconds.

sys()

A number of sys() functions have been added, and are summarized here:

- sys(231) returns zero in headless server.
- sys(233) returns empty in headless server; it returns the title of the main Omnis application window in the full server.
- sys(234) returns a 3 column row of information containing statistics about RESTful requests to the Omnis server.
- sys(236) returns true if VCS branching is enabled. If sys(236) returns false on a branched project, the VCS will display the default branch data.

FileOps

The FileOps $readcharacter function now has an additional parameter to test if the file is UTF-8 compliant:

FileOps.$readcharacter(iEnc,&cData[,&iOff=0,iMax=0,bChkUTF8=kFalse])

- **bChkUTF8**
  If the data does not have a BOM when iEnc is kUniTypeAuto and you are reading the entire file (so iOff and iMax are both zero), use UTF-8 as the detected encoding if the data is consistent with UTF-8 encoding rules

Component Store

Adding Controls to a Form

You can now add a component to the current design window (remote form, window or report) by first selecting the component in the Component Store and then pressing Return (this is the same behaviour as double-clicking on a component in the Component Store).

Omnis Configuration

Template Configuration File

Many of the preferences and settings in the Omnis IDE and Omnis App Server are stored in a configuration file called 'config.json' which is located in the 'Studio' folder under the main Omnis folder. The config.json file, *created in the 'Studio' folder when Omnis first runs,* contains only a subset of settings needed initially for development, but there are many more settings you can add to the file to enable or configure further features in Omnis.

There is a new template 'config.json' file containing all possible sections and settings located in a new 'templates' folder in the Studio folder: you can copy sections out of this file and add them to your copy of the config.json file in the Studio folder. (Note there is a copy of the CORS configuration file 'cors.json' in the new templates folder, which you need to move into the 'Studio' folder to enable CORS: see the Web Services section for more info.)

Configuration File Methods

There are some new methods in the Omnis Preferences that allow you get and set the contents of the Omnis configuration file. These would allow you, for example, to create your own config.json from code which could be used for deployment of your app.

- $getconfigjson()
  Returns config.json as a row (empty if config.json could not be parsed)
- $setconfigjson(wConfigJson)
  Sets config.json to the supplied row

These are methods of $root.$prefs, and they appear on the Methods tab of the Property Manager when viewin the Omnis Preferences, but only when used with the Notation Inspector.

You can use them to modify existing items, or add new items. For example:

```
Do $prefs.$getconfigjson() Returns cRow

If isnull(cRow.obrowser.$cols.$findname("newitem"))
Do cRow.obrowser.$cols.$add("newitem",kCharacter,kSimplechar,1000000)
End If
Calculate cRow.obrowser.newitem as "my test2"

If isnull(cRow.obrowser.$cols.$findname("newitem2"))
Do cRow.obrowser.$cols.$add("newitem2",kBoolean)
End If
Calculate cRow.obrowser.newitem2 as kTrue

If isnull(cRow.obrowser.$cols.$findname("newitem3"))
Do cRow.obrowser.$cols.$add("newitem3",kInteger,0)
End If
Calculate cRow.obrowser.newitem3 as 123

Do $prefs.$setconfigjson(cRow)
```

VCS

VCS Branching

Access to branching in the VCS has been removed from the Studio Browser but support for branching is still available for backwards compatibility. To enable branching, you can add the following member to config.json:

```
"vcs": {
    "enableBranching": true
}
```

In addition, sys(236) has been added and returns true if VCS branching is enabled. If sys(236) returns false on a branched project, the VCS will display the default branch data.

Showing Checked Out Classes

There is a new hyperlink option in the Libraries view of the Studio Browser “Show Checked Out” to display only checked out classes in the current library. Once enabled you can click the option again to show all classes.

Checking Out Classes

There is a new option on the Check Out tab in the VCS Options: when enabled, the **Ignore Checked Out Classes** option ensures that the VCS will not copy out a component if it is already checked out.

Window Components

Combo Boxes

Combo boxes in window class toolbars now support $keyevents, such as evKey.

OJSON

Static Methods

The OJSON object has two new static methods: $listtoarrayarray and $arrayarraytolist, to manipulate an array of arrays.

- **$listtoarrayarray()**
  OJSON.$listtoarrayarray(lList[,iEncoding=kUniTypeUTF8,&cErrorText]) writes a list with simple columns to an array of arrays; returns the JSON with specified encoding (UTF8, UTF16BE/LE, UTF32BE/LE or Character). Returns NULL and cErrorText for an error.
- **$arrayarraytolist()**
  OJSON.$objectarraytolist(vData[,&cErrorText]) parses vData (binary (UTF8/16/32) or character). vData must be a JSON array of arrays containing members with simple types. Returns a list representing the JSON. Returns NULL and cErrorText for an error.

These methods only work with simple types as array members. $arrayarraytolist requires that each JSON array must have the same number of elements, and each JSON array member at a particular index must be of the same data type.

XML

Using Schema Files for Validation

Some new properties have been added to the XML DOM Document object to allow you to use an external schema file (XSD) for validation.

- $fullschemavalidation
  If true, and $parservalidates is also true, and the parser will validate against a schema, the parser performs additional checks against the schema.

You should set $fullschemavalidation to true unless performance is an issue.

The other new properties allow an external schema to be specified:

- $nonamespaceschemalocation
  if specified, this property becomes the noNamespaceSchemaLocation attribute for the document being parsed.
- $schemalocation
  if specified, this property becomes the schemaLocation attribute for the document being parsed.

The schemas specified in these properties need to be referenced by a pathname to the schema file.

For example, to use an external schema, turn on $fullvalidation (without this, the absence of the schema file is an unreported and ignored warning), and set $schemalocation to:

```
"urn:books c:\dev\studio60orfc\oxml\test\books.xsd"
```

where the second component is the path to the schema file on your system.

If an XSD is in the same directory as the XML, you can use:

```
"urn:books books.xsd"
```

Localization

String Table Editor

The String Table Editor now saves to Tab Separated Value (.tsv) format by default, but provides the option to save in the old .stb format. In addition, the dialog for opening a string table initially defaults to .tsv.

Commands

Text: and Sta: Commands

The *Text:* and *Sta:* commands now enclose their parameters in curly braces {}. This change was made in order to allow chroma coding and construct highlighting in the Method Editor to work consistently, and for the new Library (JSON) export option to work for these commands.

Web Server Plugins

VC++ Runtime Library

The dependency of the Omnis web server plugins on the VC++ runtime library, available separately from Microsoft, has changed. The plugins omnisapi.dll, nph-omniscgi.exe, and their Web Services (REST) variants are now statically linked to the VC++ runtime redistributable library. As a result, weshared, apache mod_omnis and omnisservlet are also statically linked to the library.

SQL Query Builder

Some enhancements have been added to the SQL Query Builder (added in Studio 8.1.1), which is available in the SQL Browser inside the Studio Browser.

A 'Create table class' option has been added to a new 'Other' toolbar menu option for creating a table class from the current query; the option creates a $load method in the table class contains the query from the Query Builder. The option also gives you the option to create a window class and/or a remote form for viewing the data via the new table class; the form contains code which calls the $load method in the table class.

An 'Export Data' option has been added to the 'Other' toolbar menu to allow you to export the results data.

Plus the 'Create Statement on Clipboard' option has been added to the 'Other' menu option; the Omnis code generated by this option is suitable for pasting into an Omnis method.

What's New in Omnis Studio 8.0.3

The following enhancements have been added to Omnis Studio 8.0.3:

- **SQLite Encryption**
  the SQLite DAM now supports native datafile encryption: when enabled, data is encrypted and can only be read and decrypted using the encryption key
- **Dictation**
  allows end users to enter text into Edit fields using the built-in dictation on macOS Sierra; dictation must be enabled in the config.json file
- **Apple Events**
  a new object class containing AppleScript to run various Apple Finder events, to replace the Apple Events commands which have now been made obsolete
- **Map Markers**
  extended support for Google Maps allows you to add a larger variety of map markers (circles, arrows) and polygons to maps in JavaScript apps
- **Page Panes**
  the JS Paged Pane control has a new property $animatetransitions, which allows you to animate the transition when the current page is changed
- **Worker Objects**
  additional support for notifications in the Worker Objects, for example, to allow you to report progress on a long operation in your SQL transactions
- **JSON column types in PostgreSQL DAM**
  you can select and insert JSON strings into PostgreSQL JSON and JSONB columns
- **Hardware ID**
  a new function to return the string ID of the hardware on which Omnis Studio is currently running; this replaces sys(227) which has been removed
- **Icon functions**
  There is an additional optional noscale parameter to the $getpict() and $getmask() functions in the OmnisIcn Library function group

SQLite Encryption

The SQLite DAM now supports native datafile *encryption*. When enabled, all data written to the SQLite datafile is encrypted and can only be read and decrypted using the SQLite DAM with the appropriate encryption key.

Encryption is enabled by setting the session object $encryptkey property before logging on to the SQLite datafile. $encryptkey accepts a string of hexadecimal characters. The string should be of even length and should be no longer than 32 characters. The key value will be truncated if it does not meet either of these criteria. The accepted key value is then used to seed an internal private key which is subsequently used by all statement objects belonging to that session object.

To create a new encrypted datafile, the $opencreate property should also be set to kTrue before logging on. For example:

```
Do sessObj.$opencreate.$assign(kTrue)   ;; create a new datafile if it does
   not exist
Do sessObj.$encryptkey.$assign('1a2b3c4d5e6f') Returns #F
Do sessObj.$logon('/Users/user1/Desktop/sqlite.db','','','session1') Returns
   #F
```

Once encrypted, $logon() will fail unless the correct $encryptkey is supplied. $encryptkey will be ignored (cleared) if the DAM detects a connection to a non-encrypted datafile. Please note that you cannot change the $encryptkey property while the DAM is logged on. Errors encountered during assignment of $encryptkey are written to session.$nativeerrorcode and session.$nativeerrortext.

The DAM provides two session methods that facilitate encryption/decryption of existing SQLite datafiles:

- $encrypt(filename)
  opens a non-encrypted datafile and encrypts it using the $encryptkey. A backup copy of the non-encrypted datafile is created at the file location named *filename.bak*
- $decrypt(filename)
  opens a previously encrypted datafile and decrypts it using the $encryptkey. A backup copy of the encrypted datafile is created at the file location named *filename.bak*

$encrypt() and $decrypt() return kTrue on success but will fail, unless the DAM is logged off, if the process cannot get exclusive read/write access to the specified datafile or if *filename.bak* already exists and cannot be overwritten. Once encrypted, connection via third-party tools should be avoided as this may result in undefined behaviour and cause datafile corruption.

Dictation for Edit Fields

You can now enter text into an edit field using the built-in Dictation feature on macOS, which tries to convert audible speech into meaningful text. To allow dictation to occur the focus must be in the edit field, which must itself be editable, i.e. not disabled, and dictation must be enabled on the client computer. Dictation is available in Single- and Multi-line edit fields, the edit part of Combo boxes, and edit fields in Complex grids in remote forms (and window classes), that is, wherever text input is required.

Enabling Dictation

Support for Dictation is turned on in Omnis by default, but you can change it in the config.json file (prior to Studio 8.1 it was off by default). There is a "useDictation" option in a new "macOS" member in config.json, which is set to true to enable dictation; note you have to quit Omnis to change the config.json file, and any change will be effective when you restart Omnis.

```
"macOS": {
        "useDictation": true
    }
```

Using Dictation in Edit fields

To enter dictation mode, place the cursor in the edit field and select the *Start Dictation* option from the Edit menu on macOS, or press the Function key twice (Fn + Fn). This will open the dictation popup (usually at the insertion point, or in the center of the screen) and put the computer in listening mode. Dictation can be stopped or cancelled by clicking on Done in the popup, or using the *Stop Dictation* menu option.

Dictation Levels

There are two levels of dictation provided by macOS: *Standard* or *Enhanced*. These can be enabled from System Preferences->Keyboard->Dictation, or on older systems System Preferences->Dictation & Speech.

Standard dictation (the default) requires an internet connection and provides speech to text translation using Apple's servers. On older systems, the text is not translated until the Done button is pressed on the popup. On newer systems text is translated and placed into the field while the end user is speaking. Dictation will end automatically when text is entered from the keyboard or the field loses the focus.

Enhanced dictation requires the enhanced dictation engine to be downloaded, which is approximately 500MB for each language pack. This will then provide local machine based translation. Features of enhanced dictation are live feedback and offline support. With live feedback the text is rewritten while speaking. Enhanced dictation also provides spoken dictation commands such as "Select All", "Cut that", "Move left", and so on. When enhanced dictation has been started it is possible to change the currently focused edit field and move the popup to the new field and continue to dictate. It is also possible to type and dictate at the same time.

Apple Events

All the Apple Event commands, including *Send core event* and *Send Finder event,* have been made obsolete in this version; they do not work on macOS Sierra and are therefore no longer supported in this release of Omnis Studio. The commands have been moved from the Apple events… group and placed into the *Obsolete* commands group in the Method Editor.

Apple Events Object

To replace the functionality of the old “Send Finder Event” commands, this release includes a new Object class called **oFinderEvent** which contains a number of methods which run AppleScript to execute the equivalent Apple Finder events, such as a *Get File Info* event or a *Duplicate Files* event. The AppleScript is run using the Omnis $runapplescript() method from inside each method in the object class.

To use the object class and these methods, click on the Class Wizard option in the Studio Browser, then click on Object, select the oFinderEvent option, name the object class (or keep the name oFinderEvent) and press Return: a copy of the object class template is added to your library. Open the Method Editor for the class in which you want to use the Finder events (such as a window, menu or toolbar class), and then create an **Object variable** in the class, setting its subtype to the oFinderEvent object you created.

Apple Event Methods

You can call the methods in your code, and run the AppleScript as required, using the Omnis command *Do YourObjectVar.$methodname()* using the appropriate method name, as below.

Some of the methods can take a file path as the first parameter, or if this is omitted or empty a file selection dialog will open. The title of the dialog can be customized by editing the cOpenFilesTitle class variable.

- **$getfileinfo**([cFilePath])
  Sends a Get File Info event: equivalent *Send finder event {Get File Info}* command
- **$duplicatefiles**([cFilePath])
  Sends a Duplicate Files event: equivalent *Send finder event {Duplicate Files}* command
- **$makealiasforfiles**([cFilePath])
  Sends a Make Alias For Files event: equivalent *Send finder event {Make Alias For Files}* command
- **$openfiles**([cFilePath])
  Sends a Open Files event: equivalent *Send finder event {Open Files}* command
- **$printfiles**([cFilePath])
  Sends a Print Files event: equivalent *Send finder event {Print Files}* command
- **$revealfiles**([cFilePath])
  Sends a Reveal Files event: equivalent *Send finder event {Reveal Files}* command
- **$emptytrash**()
  Sends a Empty Trash event: equivalent *Send finder event {Empty Trash}* command
- $**restart**()
  Sends a Restart Macintosh event: equivalent *Send finder event {Restart Macintosh}* command
- **$shutdown**()
  Sends a Shutdown Macintosh event: equivalent *Send finder event {Shutdown Macintosh}* command

- **$sleep**()
  Sends a Sleep Macintosh event: equivalent *Send finder event {Sleep Macintosh}* command

The object has three instance variables which you can use in your code to handle errors:

- **iErrCode**
  The error code generated by the last command. 0 for no error.
- **iErrText**
  The error text generated by the last command.
- **iScript**
  The AppleScript sent by the last command.

The following legacy commands are not supported in the latest version on macOS: Send finder event {Show About}, Send finder event {Share Files}, Send finder event {Show Clipboard}.

You can examine the Omnis code and AppleScript in each method inside the object class. For example, various simple operations are handled in a generic method $simpleop and the operation is passed in as a parameter:

```
; $simpleop method
; pOperation param receives 'Empty', 'Restart', 'Shut down', or 'Sleep' msg
Begin text block
Text: tell application "Finder" (Carriage return)
Text: [pOperation] (Carriage return)
Text: end tell (Carriage return)
End text block
Get text block iScript

Do $root.$runapplescript(iScript,iErrCode,iErrText)
Quit method iErrCode
```

Each of the new methods in the object class includes the equivalent old command as a comment to help you map your code to the new methods.

```
;  Send finder event {Empty Trash}  ;; old command
Quit method $cinst.$simpleop("Empty")  ;; new method
```

Map Control

Custom Markers

You can add your own icon to markers in the JavaScript Map control, by assigning an icon URL in the fifth column of the map marker list for the control – if the fifth parameter was omitted the default Google map marker icon is used: this feature was available in previous versions. It is now possible to assign an alternative marker icon or symbol, including map markers from the Google maps API, by adding a sixth column to the marker list: in this case the fifth column should be omitted.

The definition for the markers list in the JavaScript Map control can now be:

```
Do iMapMarkers.$define(
   iMarkerLatLong,iMarkerTitle,iMarkerTag,iMarkerHtml,iMarkerIcon,iMarkerCustom
   )
```

where *iMarkerCustom* is a new string column (column 6) specifying a custom marker. When a marker is defined in the marker list, and the iMarkerIcon (column 5) is empty, iMarkerCustom can be included with the following attributes, separated with a '|' character (you only need to specify the attributes required). An example custom string would be:

```
"path:google.maps.SymbolPath.BACKWARD_CLOSED_ARROW | fillColor: red |
   fillOpacity:0.8 | scale: 4 | strokeColor:black | strokeWeight: 1"
```

Or to draw a five-pointed star marker:

```
"path:M 125,5 155,90 245,90 175,145 200,230 125,180 50,230 75,145 5,90 95,90 z
   | fillColor: red | fillOpacity:0.8 |  scale: 0.1|strokeColor:black |
   strokeWeight: 1 | anchor:122,115"
```

Or to draw a circle marker:

```
"path:google.maps.SymbolPath.CIRCLE | fillColor: red | fillOpacity:0.8 |
   scale: 4"
```

Where the custom marker parameters are defined as:

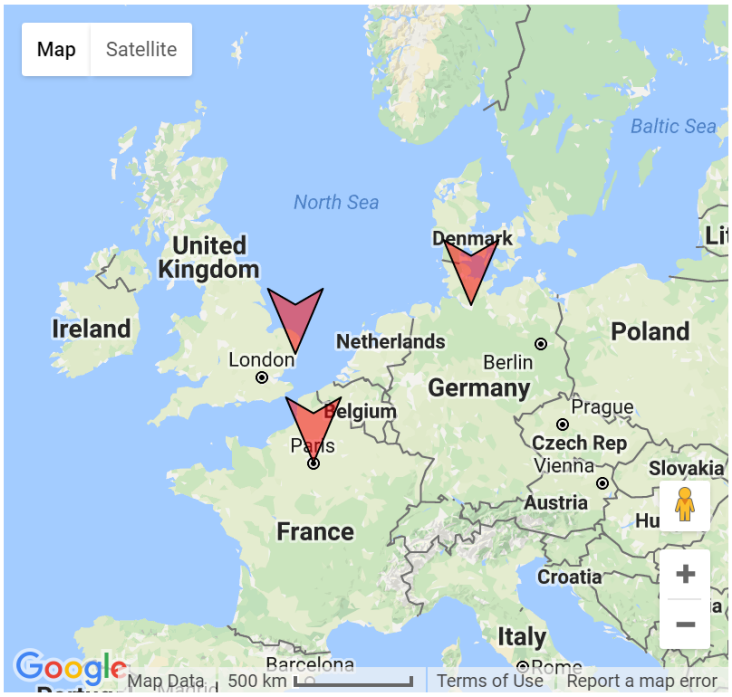
- **path** can either be a map symbol, or an SVG notation path, as defined below
- **fillColor** the color used to fill the marker object, an html css color name or value e.g. #FF0000
- **fillOpacity** the opacity of the fill color, a value from 0 to 1, e.g. 0.5 is 50% transparent fill
- **scale** a scaling factor for the object
- **strokeColor** the color used to outline the object, an html css color name or value e.g. #FF0000
- **strokeWeight** the thickness of the stroke line
- **anchor** allows you to set the anchor position or offset the shape. By default, shapes are aligned to the top left of the marker relative to its lat:long

Marker Symbol – prefixed google.maps.SymbolPath.	**Description**	
CIRCLE	A circle.	
BACKWARD_CLOSED_ARROW	A backward-pointing arrow that is closed on all sides.	
FORWARD_CLOSED_ARROW	A forward-pointing arrow that is closed on all sides.	
BACKWARD_OPEN_ARROW	A backward-pointing arrow that is open on one side.	
FORWARD_OPEN_ARROW	A forward-pointing arrow that is open on one side.	

For example:

```
Do iMapMarkers.$define(
   iMarkerLatLong,iMarkerTitle,iMarkerTag,iMarkerHtml, ,iMarkerCustom)
Do iMapMarkers.$add(
   "52.223460:1.492379","Omnis UK","Omnis
   UK","","","path:google.maps.SymbolPath.BACKWARD_CLOSED_ARROW|fillColor:
   red|fillOpacity:0.8| scale: 8|strokeColor:black|strokeWeight: 1")
   ; the JS Map app uses similar code to show the Omnis offices
```

The JS Map example app has been updated and includes some of the new markers and polygons, and can viewed or downloaded via the Omnis website (www.omnis.net) from the JavaScript Component Gallery. The following image shows the location of the European Omnis offices using the “Backward-pointing Closed Arrow”.



The map control now has a property $fitmaptomarkers that can be assigned value 1 at runtime to force the map to zoom in or out to allow all the map markers to be shown.

Finding the Latitude:Longitude

To find the lat:long position of somewhere (to be used in the JS Map control in the iMarkerLatLong parameter), you can Right-click somewhere on a Google map in a standard browser (not the Omnis JS Map control), select the 'What's here' option and the latitude:longitude value of that position is shown on the popup. You need to replace the comma with a colon to be used as a parameter in Omnis, e.g. 52.223460:1.492379.

Polygon Objects

In addition to icons and standard map markers, you can add polygon objects or irregular shapes to maps in the JavaScript Map control. The new property $mappolys specifes the data name of a list variable which contains the definition of each polygon or shape as follows:

```
Do iPolyMarkers.$define(
   iPolyLatLong,iPolyStroke,iPolyOpacity,iPolyWeight,iPolyFill,iPolyFillOpacity
   ,iPolyTag)
```

- **iPolyLatLong** the latitude:longitude values for each of the points of the polygon, so a triangle would have 3 points: the lat:long settings are separated with the '|' character, e.g. 25.774,-80.190|18.466,-66.118|32.321,-64.757|25.774,-80.190
- **iPolyStroke** the color used to outline the polygon, which is an html css color name or value e.g. #FF0000
- **iPolyOpacity** the opacity of the stroke color, a value from 0 to 1, e.g. 0.5 is 50% transparent
- **iPolyWeight** the thickness of the stroke line
- **iPolyFill** the fill color of the polygon object, an html css color name or value e.g. #FF0000
- **iPolyFillOpacity** the opacity of the fill color, a value from 0 to 1, e.g. 0.5 is 50% transparent
- **iPolyTag** the tag name or label for the polygon, which is sent to the evPolygonClicked event method in pPoly

For example, the following code draws the Bermuda Triangle on the map (see the JS Map example app):

```
Do iPolyMarkers.$add(
   "25.774,-80.190|18.466,-66.118|32.321,-64.757|25.774,-
   80.190","#FF0000","0.8","3","#FF0000","0.35","Bermuda Triangle")
```

There is a new event evPolygonClicked with the parameter pPoly, which is called when a polygon on the map is clicked, and pPoly will be set to the polygon tag as defined in the list.

Paged Panes

Animated Transitions

The JavaScript Paged Pane control has a new property $animatetransitions. If enabled, the transition between pages will be animated when the current page is changed. The property cannot be changed at runtime (the same as $scrolltochangepage).

If used in conjunction with $scrolltochangepage, when the user stops scrolling, the pane will smoothly animate into position, rather than jumping instantly.

The animation time is set to 500ms, which should be fine for most purposes, but if you wish to change this, you can use JavaScript to change the Paged Pane control's (or its prototype's) ANIMATION_TIME property.

Worker Objects

Push Notifications

The worker objects allow you to push additional notifications before calling $completed. This enhancement means workers could periodically tell the application what is going on during their execution, for example, the worker could report progress to say how much of a large file transfer has completed.

PostgreSQL

JSON column types

You can select and insert JSON strings into PostgreSQL JSON and JSONB columns.

The client library will parse and validate text before insertion into JSON/JSONB columns, and you can optionally use this feature to validate JSON strings before insertion, for example:

```
Do cStat.$execdirect("select '{""col1"":1,""col2"":""600
   meters"",""col3"":[[""one"",""two"",""three""]}'::json as myValue") Returns
   #F
Do cStat.$fetchinto(lResult)    ;; returns  {"col1":1,"col2":"600
   meters","col3":["one","two","three"]}
Do cStat.$execdirect('create table jsontest(col1 int, col2 jsonb)') Returns #F
Do cStat.$execdirect('insert into jsontest values(1,@[lResult])') Returns #F
```

You can also use the $addcustomtype() method to make $createnames() generate JSON and/or JSONB column types, for example:

```
Do cSess.$addcustomtype(1000,'JSON') Returns #F
```

Functions

Hardware ID

There is a new function in the Web commands external component OWEB.$gethardwareid() that returns a string ID that identifies the hardware on which Omnis Studio is currently running. This new function replaces sys(227) which has been removed.

Icon Functions

A noscale parameter has been added to the $getpict() and $getmask() functions in the OmnisIcn Library function group.

- OmnisIcn Library.$getpict(iconid,backgroundcolor[,*noscale*])
- OmnisIcn Library.$getmask(iconid[,*noscale*])

The default is kFalse, when the parameter is omitted, which will return an image scaled to match the resolution of the current or main display. When noscale is passed as kTrue, a standard resolution image is returned.

sys(234) function

The sys(234) function has been added, which returns a row of information containing statistics about RESTful requests to the Omnis server. The row has three columns: column 1 is the count of successful calls; column 2 is count of calls resulting in an error; and column 3 is the count of calls internally re-queued because there was not a free user.

Native Switch

The text displayed on the Native Switch JavaScript component is now controlled by two new members in the built-in strings object jOmnisStrings in the JavaScript Client. You can use the new members "switch_on" and "switch_off" to replace the default text with your own text, for example, if you wish to provide different language equivalents to the default text.

Information about how to change or localise these strings can be found in the *Creating Web & Mobile Apps* manual, under "Localizing Built-in Strings".

If you wish to override the base text, you could use the following code in a separate JavaScript file loaded in your form's html file after omjsclnt.js:

```
jOmnisStrings.base.switch_on = "I";
jOmnisStrings.base.switch_off = "0";
```

Window Classes

Debugging code in oBrowser

In order to debug code running in the oBrowser component under Windows, you have to set various security settings to allow debug mode. In previous versions, these settings had to be passed on the command line as you started Omnis. In this version, you can set the security settings in the Omnis config.json file, and start Omnis normally (and if you pass the parameters on the Omnis command line now, they are ignored).

You can add the following entry to the 'obrowser' section of config.json:

```
"cefSwitches": [
    "allow-file-access-from-files",
    "disable-web-security"
]
```

The entry is an array of actual switch values to be passed to the Chromium Embedded Framework.

What's New in Omnis Studio 8.0.2

The following enhancements were added to Omnis Studio 8.0.2:

- **Mobile App Deployment**
  a new wrapper to create and deploy standalone mobile apps to run on Windows 10 based devices including desktop PCs, Surface® tablets, and Windows Phones®: in addition, the Sync Server now uses a RESTful interface to allow the Omnis Server to communicate with mobile clients
- **Custom Loading Indicator**
  there is a new client command, *showloadingoverlay,* that allows you to add a loading indicator over an individual control, or the entire page in the JavaScript Client
- **Rich Text Editor Control**
  has been enhanced and includes Code Blocks with syntax highlighting, Undo/Redo shortcut keys, Sub/Superscript, In/Outdent, Block Quotes, Clear formatting, Image uploads, Content tips
- **Worker Objects**
  Worker Objects now support an alternative completion model whereby $completed and $cancelled methods can optionally be sent directly to another instance
- **Web Services**
  support for ISO8601 based date and date-time values has been added to REST-based Web Services support; in addition, the CORS settings are now stored in a separate config file CORS.json
- **Method Templates**
  when adding the $sqldone method in the method editor Omnis now adds pre-defined or *boilerplate code* automatically, which you can add to or amend as you wish; there is a new class rfMethodTemplates in the Component Store containing method templates
- **Creating Unrecognized Variables**
  when Omnis encounters an unrecognized variable, a dialog pops up to allow you to create the variable, including options for the scope, type, subtype, etc.
- **List Variable Values**
  the Value option on the context menu for List variables in the Method Editor has some enhancements, including showing the value of $line, and the line numbers of up to the first 5 selected lines
- **Adding Blank Method Lines**
  there is a new command and keyboard shortcut on the Modify menu in the Method Editor that adds a set of blank lines to the end of the current method and sets the current line to the first new blank line
- **Sorting Variables**
  there is a Sort Names option on the View menu of the variables list window, available by right-clicking on a variable when inspecting variables in the Method Editor
- **Date and Number Formatting Override**
  entries in the system tables for storing date formats (#DFORMS), input masks (#MASKS), and so on, can now be overridden at runtime by creating and calling a configuration file called tables.json

- **Text Escapes for URIs**
  there are two new static functions, in the OWEB component, that escape text for use in URIs: $makeuri() returns a properly formed URI, and $escapeuritext() escapes a text URI
- **Generating UUIDs**
  there is a new function, in the OWEB component, $makeuuid(), that allows you to generate a new UUID

Mobile App Deployment

Windows 10 Wrapper

There is a new wrapper for deploying standalone apps on **Windows 10** devices, including desktop PCs, Surface® tablets, and Windows Phones®. The *Application Wrappers* provided with Omnis Studio 8, available for iOS and Android based mobile devices, allow you to deploy your JavaScript Client based apps as **standalone mobile apps** (rather than deployed to end users' web browsers). The wrapper SDKs are available to download from our website, together with complete documentation describing how you can build and deploy standalone mobile apps.

The Windows 10 wrapper is a "Universal Windows Platform" (UWP) app, and so should run on any Windows 10 based device. As with the iOS and Android wrappers, it allows you to build a branded app based on Omnis JavaScript Client remote forms, with the ability to access device functionality, local database and data synchronization features, as well as the ability to run self-contained, without the need for an Omnis App Server. This will allow you to turn your Omnis JavaScript Client application into a native look-and-feel Windows 10 application, which can be distributed within an organisation, or via the Microsoft Store.

Note: the wrappers are released separately to the main Omnis Studio SDK, so in this case the Windows 10 wrapper SDK will follow shortly after the Studio 8.0.2 release, and you will be able to download it from the Omnis website: www.omnis.net/download/jswrapper.jsp

Sync Server

There is a new version of the Omnis Sync Server, version 2.3, which uses a RESTful interface to communicate with mobile clients. Due to this change you now require a Web Services serial number to use the Sync Server in the Development version of Studio 8.0.2 (all Server versions of Omnis include Web Services support). Contact your local sales office for further information.

$syncinit HostString

This change necessitates a small breaking change to libraries which synchronize with the Sync Server, in that the format of the **HostString** passed to the $syncinit method has changed:

For a direct connection to the built-in Omnis Server, the *HostString* should be:

```
http://<ipaddress>:<$serverport>
```

If you are connecting through a web server, you need to add the **omnisrest**… server plugin to your web server, in the same way as the other server plugins described in Tech Note: TNJS0003, and connect through that.

The *HostString* should then be of the form:

```
http://<web server address>/<Omnis rest plugin>/ws/<XXX>
```

Where <XXX> is either:

- <Omnis $serverport> (if Omnis is on the same machine as the web server)
- <Omnis server ipaddress>_<Omnis $serverport>
- <Server Pool>_<Omnis server ipaddress>_<Omnis $serverport>

For example:

```
http://mysite.com/cgibin/omnisrestisapi.dll/ws/192.168.1.14_7001
```

Wrapper Compatibility

Due to the addition of RESTful support in the Sync Server, new application wrapper apps are required if you are using the new Sync Server. Version 1.5.0 of the wrappers

and later are compatible with the new Sync Server, and these can be downloaded from the Omnis website: www.omnis.net/download/jswrapper.jsp

JavaScript Client

Custom Loading Indicator

You can now add a loading indicator (animated image) over an individual control, or the entire page in the JavaScript Client, using a new client command, *showloadingoverlay*. The new loader provides feedback to the user, that a long running operation may be in progress, and it will also prevent user input. It is useful if you are doing any asynchronous operations, such as populating a long list using a SQL worker object. Client commands, such as showloadingoverlay, are executed using the $clientcommand method, as follows:

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

Where *rowvariable* is row(show*, controlName, message, cssClass*).

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

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

The loading overlay comprises a toplevel div, which will be given your CSS class name. This contains a div with the CSS class name of "**container**", which contains a div with the "**indicator**" class and a 'p' element with the "**message**" class. You will need to use CSS to style all of these.

The omnisLoadingOverlay class in omnis.css may be useful as a basis.

Rich Text Editor Control

The Rich Text Editor is now based on Quill 1.0, which provides several new features and enhancements, including the ability to display Code Blocks with syntax highlighting, Undo/Redo shortcut keys, Sub/Superscript, In/Outdent, Block Quotes, Image uploads, Content tips, and so on. The component in previous versions of Omnis Studio was based on a previous release of Quill.

Dynamically Loaded Resources

The Rich Text Control now dynamically loads the necessary JavaScript & CSS files when it is used. To this end, you will find that there are several new files in the Studio tree:

```
html/scripts/
   quill.js
   quill-legacy.js
   highlight.pack.js
html/css/
   quill.snow.css
   quill-legacy.snow.css
   highlight-theme.css
```

When deploying to a web server, you must make sure to also copy these files over.

Quill 1.0 only supports IE 11 or above. When running on earlier IE versions, the control will fall back to Quill 0.2 - this is what the '-legacy' files above are used for.

Code Blocks

The updated Rich Text Editor allows you to insert **Code Blocks**. These allow you to insert syntax-highlighted code. The syntax highlighting is achieved using highlight.js and by default includes highlight support for several popular languages.

If the language(s) you require is/are not supported out of the box, you can create a 'Custom Package' on the highlight.js download page, and replace the **highlight.pack.js** in your tree/web server with the one you download.

Similarly, if you want to change the code block's appearance, you can take any of the theme css files from your highlight.js download, rename it **highlight-theme.css** and replace the supplied file with your own.

Other Additions

- High-res Toolbar Icons
- You can Undo/Redo your recent changes to the editor using Ctrl/Cmd + Z to Undo or Ctrl/Cmd + Y to Redo
- Subscript & Superscript
- Indent & Outdent
- Embed Videos
- Block Quotes
- Clear Formatting
- Images can now be uploaded from the client
- A content tip can be assigned

New Properties

$contenttip: Allows you to specify some text to be displayed in the editor when it has no content.

$removedtoolbaritems: A bitmask of kJSRichText… values, allowing you to specify toolbar items to hide in your Rich Text Editor instance.

New String Table IDs

The following additional string table IDs may be specified, to localize/customize the text displayed in the Rich Text editor & its tooltips:

Tooltips for buttons

rt_decrease_indent	rt_increase_indent	rt_video
rt_blockquote	rt_codeblock	rt_clearformat

Text Displayed On Controls

rt_sansserif	rt_serif	rt_monospace

Component Icons

The ‘Studio’ icon set, added for Studio 8.0.1 and located in the ‘iconsets’ folder, has been added to the search path for component icons. When a reference to an icon is made in a component (e.g. in the $iconid property for a button), libraries now search the ‘Studio’ icon set automatically, in addition to their own icon set specified in $iconset, and other locations for icons. The complete search order for icons used in a developer library is now:

1. The custom icon set for the library, under the ‘iconsets’ folder and specified in $iconset
2. #ICONS for the library, if used
3. User icon datafiles (.df1 icon files other than Omnispic and Userpic), if used
4. The ‘Studio’ icon set, under the ‘iconsets’ folder
5. Omnispic.df1 and Userpic.df1 icon datafiles

The Select Icon dialog, opened by clicking on $iconid in the Property Manager when selecting an icon for a component, now shows both the ‘Studio’ icon set and the icon set in $iconset for your library, if specified.

Existing users should note that the /html/icons folder is still supported for the location of icon sets, but going forward all new icon sets should be placed in the /iconsets folder under the main Omnis folder.

Server Date and Time Setting

There is a new entry supported in the "server" section of Omnis configuration file (config.json), "timeOffsetMinutes", to allow you to add an offset to the date-time setting on the Omnis App Server. Omnis adds the value of this setting to the current system date-time when generating the value for #D and #T. If the entry is not present, it defaults to zero, maintaining the current behavior, i.e. no date-time offset is applied.

Subform Sets

Subform Set form parameters are now passed to $init as well as $construct. This enables you to pass parameters to a form in a subform set when it is running in the serverless client, since $construct is not executed in serverless client mode, but $init is run.

Subform Instance Parameters

Omnis is now stricter on the format of the comma-separated list you pass to a subform, but this provides more consistent handling of the parameters when they are passed through to $init.

String values for parameters in $userinfo must now be enclosed in quotes (single or double), and can now contain spaces and commas. Numeric values should now be passed through as numeric values.

Device Control

The GPS and Vibrate actions in the Device control will now attempt to work outside the wrappers, if the browser supports the necessary APIs. Even though most browsers do support the APIs, you should not rely on them being available and you should test your apps thoroughly to verify the behavior of these actions in your app.

Component Borders

The borders of JavaScript components specified by the kCtrlBorderShadow... constant now draw *within the bounds of the control,* for both Windows and macOS, and have the same dimensions for both platforms. This may have a small, but significant effect regarding the apparent spacing of the controls in remote forms.

JavaScript & External Component Icons

The JavaScript Components are implemented as external components in Omnis Studio, and they are listed in the Component Store under the **JavaScript Components** tab (press F3 while editing a remote form to show the components). In this version, the JavaScript Components can be shown using Large Icons.

All external components now support high-definition icon images, which can be added as separate PNG image files; in previous versions, 16x16 was the maximum size allowed for external components and alpha images were not supported.

Worker Objects

Worker Objects now support an alternative completion model. The $completed and $cancelled methods can optionally be sent directly to another instance. This means you do not need to sub-class the worker object, in order to receive its results. We would recommend that you use object references rather than objects for this technique.

In order to use this new functionality, there is a new property of worker object instances, called $callbackinst. If you do not use this new property, behavior is unchanged from Studio 8.0.1 and earlier.

For example, if iHttp is an HTTP worker (an instance variable in a window class), then within the window instance you can execute:

```
iHttp.$callbackinst.$assign($cinst)
```

You need to implement $completed and $cancelled in the window class methods. The parameters are as follows:

- $completed(*row,object*)
  where *row* is a parameter of type Row, same definition as that passed to $completed in the sub-classed object when not using $callbackinst.
  object is a parameter of type Object reference (when the worker object is an object reference) or Item reference (when the worker object is an object). object is the worker object for which $completed is being called.
- $cancelled(*object*)
  where *object* is the same as for $completed

Web Services

Date and Date-time values

Support for 'date' and 'date-time' values has been added to REST-based Web Services support in Omnis. RESTful services typically use a subset of ISO8601 to exchange

date and date-time values, which are supported in Swagger which is used to define web services in Omnis. ISO8601 represents the date or date-time as a character string. See http://swagger.io/specification/ and search for RFC3339 in the page - a link from there takes you to http://xml2rfc.ietf.org/public/rfc/html/rfc3339.html#anchor14.

Swagger has two format specifiers for character string values: date and date-time. The Swagger generator has been enhanced so that for fields (either in a schema or in the HTTP method parameters) of type character, the description can contain a swagger tag that specifies the format, e.g. the description for query string parameter startDate could be “<swagger-date> This parameter is the start date for the requested work”. When Omnis generates the swagger definition for the web service, it looks for these swagger tags, and uses them to set the format for Swagger string types. This has the additional benefit that you can use other supported Swagger string formats, e.g. password and byte.

ISO8601 date functions

Dates and date-time values are still exchanged as character values. Your application code therefore needs to parse and generate the ISO8601 date and date-time values. To support this, there are two new functions to manipulate ISO8601 dates, or at least the subset of ISO8601 needed to work with Swagger and the Omnis RESTful server:

- **iso8601toomnis()**
  iso8601toomnis(cISO8601,bNeedTime,bHasTimeZone,cErrorText]) converts ISO8601 date/date-time string to Omnis date-time and returns result (in UTC time if cISO8601 contains time and time zone).
  Returns #NULL and sets cErrorText if an error occurs
- **omnistoiso8601()**
  omnistoiso8601(dOmnisDateTime,bNeedTime[,cErrorText]) converts dOmnisDateTime (assumed to be in UTC) to an ISO8601 date or date-time string (depending on bNeedTime) and returns the result.
  Returns #NULL and sets cErrorText if an error occurs.
  Hundredths are always rounded down to the nearest tenth.

Note that for a RESTful service, you should always use time zones for input date time values, so you would always pass bHasTimeZone as kTrue to iso8601toomnis() if you are passing bNeedTime as kTrue.

omnistoiso8601() always outputs the timezone using the “Z” UTC time indicator.

CORS

The configuration of CORS for RESTful-based web services is now stored in a separate configuration file, cors.json in the Studio folder: if the cors.json file is not present in the Studio folder, then there is no configuration setup for CORS. In previous versions, the CORS configuration was stored in a “CORS” section in studio.json file which is now redundant. Since it is a separate file, you can edit it while the Omnis Server is running. (Note from Studio 8.1 onwards a template cors.json file is located in the Studio/config folder, and you need to move this file into the Studio folder to enable CORS.)

The cors.json file has the same syntax as the “CORS” member in old studio.json file. The content of cors.json would be something like this:

```
{
    "originLists": {
        "list1": [
            "http://127.0.0.2",
            "http://127.0.0.2:8081",
            "http://localhost1:8081",
            "http://online.swagger.io"
        ]
```

```
    },
    "APIS": {
        "*": {
            "origins": "list1",
            "headers": "*",
            "supportsCredentials": true,
            "maxAge": 0
        },
        "Swagger": {
            "origins": "list1",
            "headers": "*",
            "supportsCredentials": true,
            "maxAge": 0
        }
    }
}
```

As this is a separate file, it can be changed while the Omnis Server is open, but you need to inform Omnis of the change. This can be done by clicking the "Reload CORS Config" button on the Server Configuration dialog, which can be loaded from the Studio Browser when the Web Service Server option is selected.

Method Editor

Method Templates

When you add the $sqldone method to a remote form, Omnis adds pre-defined or boilerplate code, as well as the required parameter variables, and sets the method to execute on the client automatically. This saves you having to add the same code every time you want to create the $sqldone method – you can then add to or amend the code as you wish.

When you add $sqldone (which is the client-executed completion method for SQL objects), Omnis will add a set of pre-defined lines of code and the required parameter variables. In this case, the code added to $sqldone is:

```
;; parameter vars pRequestId (Var type), pList (List) created
;; local vars lErrorCode and lErrorText created
;; Check for an error:
    Do $cinst.$sqlobject.$getlasterrorcode() Returns lErrorCode
    If lErrorCode<>0     ;; sql error occurred, show message
        Do $cinst.$sqlobject.$getlasterrortext() Returns lErrorText
        Do $cinst.$showmessage(lErrorText,'SQL Error')
        Quit method
    End If

    Switch pRequestId
    ;; Add cases for the IDs returned by your requests here.
    End Switch
```

The added code first checks if there was an error, then creates a Switch statement to handle the results based on the request in pRequestId. If you do not want to use this code, just select the lines of code and delete them.

Note this is an extension of a feature in Studio 8.0, where some methods are marked as client executed automatically when the method is added, and any parameters are added to the method if required. The methods that Omnis inserts as client executed automatically are: $candrop, $drag, $filereadcomplete, $init, $term, $getscrolltip, $filtergrid, $sortgrid, $sfsorder, $sfscanclose, $pushed, and $sqldone.

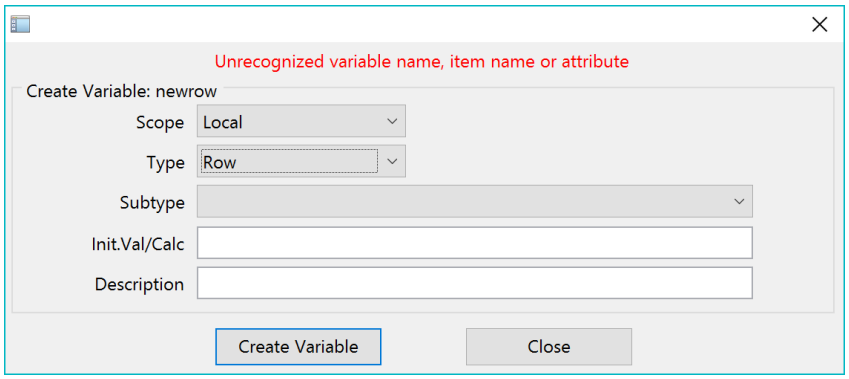
Method Template Component Store class

There is a new remote form class in the Component Store library, called **rfMethodTemplates,** which contains a method for each method template; at present there is only one method template $sqldone. To view the Component Store library and the method templates, right-click on the background of the Component Store, select the 'Show Component Library in Browser' option, and open the rfMethodTemplates class.

You can modify the methods in the rfMethodTemplates class, or add your own to implement templates for any of the following client methods: $candrop, $drag, $filereadcomplete, $init, $term, $getscrolltip, $filtergrid, $sortgrid, $sfsorder, $sfscanclose, $pushed, and $sqldone. A template method must be added as a class method.

Creating Unrecognized Variables

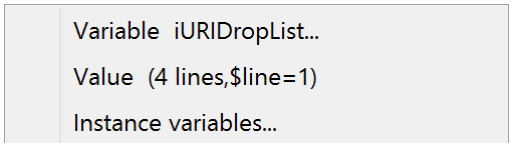
In previous versions, when Omnis encountered a variable that did not exist it popped up a warning, but no further action was taken. However, in this version a dialog will pop up automatically that allows you to create the unknown variable, including options for the scope, data type, subtype, initial value and description. The dialog only opens if the "invalid" text is a possible variable name within the current context.



The unrecognized variable dialog can open when entering code in the method editor, and when assigning a property in the property manager. In the latter case, e.g. when entering a variable name into $dataname, the initial type for the new variable is set to the most likely data type for the control – therefore, a variable for a List box control would be set to List data type. The dialog restricts the scope of the new variable to what makes sense based on class type, and so on.

List Variable Values

The **Value** option on the context menu for List variables in the Method Editor has some enhancements: this option is displayed when you right-click on a list variable name.



- The **Value** context menu option on a list variable previously showed "Value (Not Empty)" when the list contained lines. The option now tells you the number of lines in the list, the line number of the current line held in $line, and the line numbers of up to the first 5 selected lines: e.g. Value (10 lines, $line=4, $selected=1,4,8)
- When you select **Value,** the text written to the Omnis trace log includes the line number of the current line held in $line, and the line number(s) of all of the selected lines, up to the log entry limit of 255 characters (with an ellipsis at the end if necessary).

- The field value dialog has a new option "Open Lists At Current Line" which defaults to true (the state is saved with the window setup): when true, the grid opens so that the current line is visible.
- In addition, the Goto line command, on the context menu for the line numbers, sets the default line in the popup it opens to the current line.

Sorting Variables

There is a **Sort Names** command on the **View** menu of the variables list window available in the Method Editor when inspecting variables (e.g. right-click on an instance variable and choose 'Instance variables…'). The sort order is always set to an ascending sort, and is not case sensitive. This item is toggled on or off when selected (the state is saved with the window setup).

Adding Blank Method Lines

There is a new command on the **Modify** menu in the Method Editor, and on the context menu for method lines, that allows you to add a set of blank lines to the end of the current method. When the focus is on a method line, the **Append blank lines** menu option adds blank lines to the end of the current method and sets the current line to the first added blank line: you can also use the shortcut Ctrl/Cmnd-B when the focus is on the code editing area in the current method. This option behaves in the same way as clicking on the dead space at the end of a method in the method editor (shown in gray), but gives you the option to do this from the Modify menu, or from the keyboard using the shortcut key. Note that when you click away from the method, any blank lines at the end of the method are omitted automatically.

Date and Number Formatting

There are a number of system tables in Omnis that allow you to control how certain types of data is formatted and displayed: these include #BFORMS (Boolean formats), #DFORMS (date formats), #MASKS (input masks for entry fields), #NFORMS (number formats) and #TFORMS (text formats), and so on, which are stored as *system table classes* in a library (located in the 'System tables' folder in a library). You can now override individual entries within these formatting tables at runtime, without modifying the system tables in the library. This may be useful in multi-library deployments, where all of the libraries need to share the same formatting tables, but you may want to change individual settings in the formatting tables, such as the date format, according to the language or location of individual end-users.

The definitions for these alternative formatting tables can be stored in a JSON file, which should be named "tables.json" and placed in the Studio folder under the main Omnis folder. You can then use a new method called $overridetables to load an entry from the JSON file to override an entry in one of the default system tables in a library.

The override only applies while the library is open, therefore, if you close and re-open the library, you need to call $overridetables again if you still want to override the default system tables. Typically, you would do this in the $construct method in the Startup task of your library. When you call $overridetables, all design windows must be closed since they may contain controls that use the format definitions. You can close all design windows in your code using the *Close all designs* command: you need to do this before calling $overridetables to ensure all design windows are closed.

The tables.json file should contain a JSON object, and each member of the JSON object defines the content of one or more of the formatting tables: tables which do not have an entry for a member are not affected when that member is used. The following format is used:

```
{
    "en": {
        "#BFORMS": [ "[GREEN]Y;[BLUE]Y","y","Y" ],
        "#DFORMS": [ "D/M/Y H~N", "D m Y  H:N"],
        "#TFORMS": [ "@@ @@ @@ @@ @U", "'('@@@@')' @@@@@@ 'EXT'@@@"],
        "#MASKS": [ ">>###-###-###", ">>##/##/####~M/D/y~"],
        "#NFORMS": [ "0.00 E+00"]
    },
    "de": {
        "#BFORMS": [ "[GREEN]Y;[YELLOW]Y" ],
        "#DFORMS": [ "D/M/Y H~N" ]
    }
}
```

The $overridetables method has the following syntax:

- $clib.$overridetables(cJsonPath,cEntry[,&cErrorText])
  Uses member cEntry in JSON table file cJsonPath to override the system tables with entries stored in member cEntry. Returns Boolean true for success, false and cErrorText for failure

For example, you could execute the following to load tables.json from the Studio folder:

```
Do $clib.$overridetables(
   con(sys(115),"studio",pathsep(),"tables.json"),"en”,lErrorText) Returns
   bStatus
```

Locale-style names have been used, such as “en”, to indentify the members for which the tables are to be loaded, but the text could be anything to identify the set of tables to be loaded so long as you use the same name in the $overridetables() method.

$overridetables replaces the contents of the system tables with the members of the array, and sets any entries after the array members to empty. The corresponding system table class becomes read-only: you can open its class editor, but you cannot change it within the class editor.

Using the $clib.$prefs notation group for the table will change the table used at runtime, but a change made using the notation will not be saved to disk.

FileOps

Errors

There is a new method in the FileOps object, $getlasterror that allows you to report any errors from the last FileOps method that was executed:

- **$getlasterror(**[&cErrorText])
  Returns the error code from the last FileOps object method executed; also optionally populates cErrorText with a description of the error. If no error occurred, the method returns zero and the error text is empty.

Pathnames

FileOps now supports long pathnames with over 255 characters, e.g. in the $getfilename method.

Large Files

The FileOps object now supports 64-bit integers for file sizes and for offsets, etc in files.

Text Escapes for URIs

There are two new static functions, available in the Web commands external component (OWEB), that you can use to escape text for use in URIs:

- **OWEB.$makeuri()**
  OWEB.$makeuri(cURIBase,lParameters) returns the URI formed by adding the parameters in lParameters to cURIBase, escaping parameter names and values as necessary. lParameters is a list: column 1 is the parameter name and column 2 the parameter value.
- **OWEB.$escapeuritext()**
  OWEB.$escapeuritext(cTextToEscape) returns the escaped form of cTextToEscape. All characters except a-z, A-Z, 0-9, - (hyphen), . (period), _ (underscore), and ~ (tilde) are escaped as %hh. Unicode characters are the escaped form of their UTF-8 representation.

Generating UUIDs

There is a new function in the Web commands external component (OWEB), $makeuuid(), to allow you to generate a new UUID.

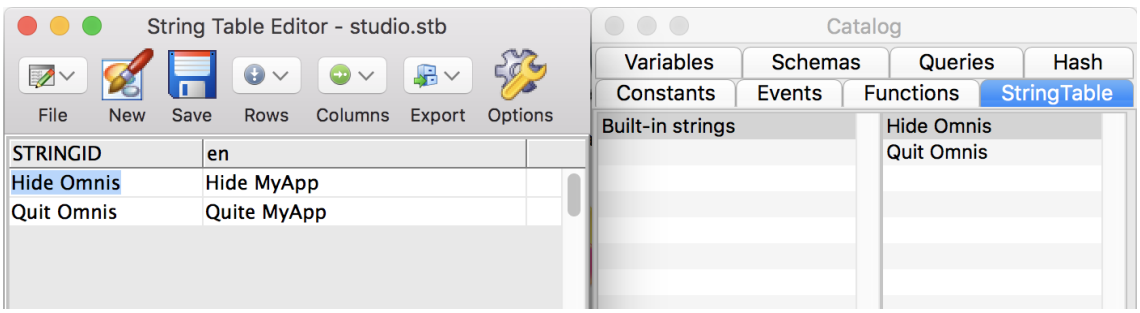
- **OWEB.$makeuuid()**
  OWEB.$makeuuid([bIncludeMinusSeparators=kTrue]) returns a new UUID as a string. bIncludeMinusSeparators affects the format of the returned string: if true, the returned UUID string is 36 characters long and includes – (hyphen) separators (so it has the standard 8-4-4-4-12 format); if false, the returned UUID string is 32 characters long and has no separators.

Deployment

Changing the Hide/Quit Omnis Option

You can now change the Hide Omnis and Quit Omnis options in the Omnis Studio runtime on macOS by adding strings to the Studio String Table (studio.stb). For example, if you have renamed the Omnis app package on macOS, to match your product name, you may also like to change the Hide Omnis and Quit Omnis options in the main application menu to reflect your product name. To do this:

- Open the Omnis Catalog (press F9) in Omnis Studio for macOS
- Select the String Tables tab and right-click on the 'Built-in strings' table (studio.stb)
- Enter the Strings "Hide Omnis" and "Quit Omnis" in the STRINGID column (without the quotes) and enter the alternative strings for each option in the 'en' column.



You can find specific strings in Omnis Studio using the Find strings… option (right-click on the string table name in the Catalog). You can drag a string from the Find window into the STRINGID column to enter the exact string to replace, and avoiding any mis-typing.

Existing users should note that the Translate button has been removed from the String Table editor since automatic translation is no longer supported.

Call DLL

Call/Register DLL

Register and Call DLL external commands now support 64-bit type specifiers.

Java Objects

Java Options

In previous versions, the $javaoptions preference was limited to 255 chars. This has been extended and the theoretical limit is now the maximum length of an Omnis character variable.

Window Classes

Debugging code in oBrowser

You can now debug code running in the oBrowser component in the Omnis Trace log, rather than in the Debug console in your browser. Console log messages sent from oBrowser now go to the Omnis trace log *by default*. You can prevent this, and use the normal JavaScript console in your browser, by setting the configuration item "useOmnisTraceLogForConsole" in the "obrowser" section of the Omnis configuration file config.json to false.

What's New in Omnis Studio 8.0.1

There were no substantive new features in Omnis Studio 8.0.1, other than new colors added to themes and an AV Player window class component for macOS only. Please see the Readme.txt file (available with the download) for any release notes and a list of faults fixed in Omnis Studio 8.0.1.

AVPlayer

There is a new external component, called AVPlayer, which allows you to play videos in your applications: note this component is only available for window classes and for macOS only. The component is available under the External Components tab in the Component Store when designing a window class on macOS. The component has the following properties and methods.

Properties

Property	Description
$allowfullscreen	If kTrue, the full screen option is enabled, in this case a button will be shown on the floating controls
$controlstyle	The type of controller shown on the player, a constant: kAVControlsNone - no controls kAVControlsInline - at the bottom of the controller kAVControlsFloating - transparent and floating over the movie (the default) kAVControlsMinimal - a minimal play back controller over the movie
$controltime	The current play time of the player in seconds
$controlvolume	The current volume of the player, 0 to 10
$videoresize	The video resize mode during playback, a constant: kAVVideoResizeAspect - video sized maintaining aspect ratio kAVVideoResizeAspectFill - video fills width or height but maintains aspect kAVVideoResizeFill - video fills the entire space

To adjust the volume of the player you could provide a slider control on your window with the following method (the $min and $max values of the slider should be 0 and 10):

```
On evNewValue
   Do $cwind.$objs.player.$controlvolume.$assign(pNewValue)
```

If you want to jump to a specific place in the movie you can set the $controltime property as follows:

```
   Do $cwind.$objs.player.$controltime.$assign(30.00) ;; 30 secs
   Do $cwind.$objs.player.$play()
```

Methods

Method	Description
$play()	plays a movie
$pause()	will pause the movie
$load(url or local path)	will load the movie

You can load a local video file to play or one from the internet using the $load() method.

```
Do $cwind.$objs.player.$load(
     "https://videos.domain.net/aboutme.mp4") Returns load_ok
```

Events

The AVPlayer component reports a single event, evAVreadyToPlay which is sent when a movie has loaded and is ready to play. You could add an $event() method to your player control with the following method to enable a Play button, so when the movie has loaded and is ready to play the end user can click the play button:

```
On evAVreadyToPlay
   Do $cwind.$objs.playbutton.$enabled.$assign(kTrue)
```

Color Themes and Appearance

Support for Themes to control colors and appearance of the IDE was added in Studio 8.0: the color settings and themes are stored in appearance.json, as well as window.json for Windows specific settings. For Studio 8.0.1, more color settings have been added for Scroll bars, Menus and Window Menu bars, the Active caption (for Windows), and the Highlight color.

Details about these new color settings have been added to this doc in the relevant section under the ‘Color Themes and Appearance’ section under Studio 8.0 new features. The additions are also listed in the Release notes section of the Readme.

What's New in Omnis Studio 8.0

The Omnis Studio 8.0 release provides 64-bit and Cocoa support for Omnis Studio running on macOS, the ability to use HTML components in window classes for Desktop Apps, Drag and Drop capability for the JavaScript Client, a new Code Assistant available in the method editor to help you write Omnis code, plus some enhancements in the Studio Browser which will help new and existing developers. Omnis Studio 8.0 includes the following features and enhancements:

- **64-bit and Cocoa support for macOS**
  The 64-bit version of Omnis Studio is now available on macOS including the SDK, App Server, and Runtime. Plus the Omnis core has been rebuilt using the Cocoa framework with anticipated benefits for speed and performance in the Omnis Studio IDE and for your macOS and iOS deployed apps
- **App Builder**
  a new tool that appears when you select New Library in the Studio Browser to help you create Omnis apps quickly and easily; the new tool provides a number of templates and steps you through the process of creating or prototyping an Omnis application, including logging onto your database, creating JavaScript forms, setting the theme, and choosing navigation for your app
- **Developer Hub**
  a new option in the Studio Browser that provides useful information for developers, such as the status of the most recent reported and fixed faults, together with information and tips for new Omnis developers; there is an Options setting in the hub to configure the content of the Studio Browser and the color theme used in the Studio IDE
- **Code Assistant and Method Editor enhancements**
  The Code Assistant is a new tool that will help you write code in an Omnis method. The code assistant will pop up automatically in the *method editor* when needed displaying command syntax and possible arguments; you can also request help using Ctrl-Space. In addition, the start and end of any block commands are highlighted (includes If, While, For, Repeat, Switch, and Begin), plus Omnis now stores a 'history list' of visited methods which you can navigate using a Back and Forward button on the method editor toolbar, and you can add notes to a method on the 'notes' tab next to the variable tabs
- **Color Themes and Appearance**
  The colors and themes used in the Studio IDE can now be changed under the Options setting in the Hub in the Studio Browser, or by changing the $appearance Omnis preference in the Property Manager: the colors used in the Studio IDE are stored in an 'appearance.json' file which you can use in your deployed apps, and you can create your own themes
- **Drag and Drop for JavaScript Client**
  provides the ability for end users to drag data between JavaScript controls in a remote form, plus end users can drag files from their desktop and drop them onto a JavaScript control within a remote form (desktop browsers only, drag and drop is not supported in mobile browsers)
- **HTML Components for Desktop Apps**
  you can enhance and enrich your desktop apps by adding HTML or JavaScript based components to thick client windows – you can create these yourself or use components available from third-parties. In addition, there is a new browser object

to allow you to embed the HTML components or present web content on your window classes

- **High Definition Displays**
  With the introduction of Retina displays on Mac desktops and laptops, and 4k displays widely becoming the standard for Windows based computers, support for high definition displays has been introduced in Omnis Studio 8.0; the Studio IDE will scale automatically if an HD monitor is detected (2x the default), and HD icons are supported in the IDE and in your own Omnis libraries using Icon sets
- **Auto Updates**
  You can now perform updates or any other changes to your Omnis application or folder structure upon restarting Omnis by adding a script to the Omnis data folder
- **Segmented Control**
  A new JavaScript control that displays a number of segments or buttons that you can use for navigation or as a toolbar within your web and mobile apps; you can assign an icon and text to each segment and you can detect which segment has been clicked
- **List Pager**
  A new property of List and Grid components to display list lines in separate pages to improve the user experience when navigating lists or grids with a large number of lines
- **Worker Objects: Push Connections**
  Support for *Push Connections* has been added to Omnis Studio to allow more control when data is returned to the client when using the Omnis worker objects, such as the SQL Worker objects

64-bit and Cocoa on macOS

In this release the Omnis core on macOS (previously named OS X) has been rebuilt using the Cocoa framework, providing added speed and performance in the Omnis Studio IDE and for your deployed applications on macOS desktops and iOS mobile devices.

Cocoa is Apple's native object-oriented API for the macOS operating system. The latest version of macOS uses the Cocoa framework as the basis of its underlying functionality as well as the user interface and overall experience. Applications built using Cocoa have a consistent look-and-feel and perform well on the latest Apple hardware including desktops and mobile devices. These improvements should be evident in the Omnis IDE and carry over into your Omnis apps on macOS and iOS.

Cocoa APIs

The new 64-bit version of the Omnis Studio core is written in Objective C++, making use of the latest Cocoa APIs and vector based drawing using Core Graphics. This allows Omnis to take advantage of newer hardware and any performance enhancements gained from the newer APIs.

The 32-bit version of Omnis Studio was a C++ Carbon application making use of legacy QuickDraw APIs for graphics rendering. A lot of these APIs have deprecated and are in danger of being removed in future releases of macOS, which is another good reason for Omnis moving to the new architecture.

HD Graphics and Fonts

Support has been added for Apple Retina (High Definition) displays which make use of high definition graphics and icons. The Omnis IDE has been re-engineered to support high definition displays and your own Omnis applications will render in high definition and will support custom HD icons. (See later in this manual regarding support for HD displays on Windows and using HD icons in your apps.)

Legacy graphics formats, such as MAC PICT are no longer supported, as PNG is now the standard shared graphics format used for icon set images.

Retina Cocoa also has improved and better looking fonts using Apple Core Text for font rendering (these are vector and not bitmap fonts).

External Components

If you create Omnis external components for macOS, and you want to use them in Omnis Studio 8.0 on macOS, you will need to convert them to use Objective C++ and the Cocoa framework: note that macOS 10.9 will be a minimum requirement to run the 64-bit Cocoa version of Omnis. An updated external component SDK will be provided with the final release that will support the majority of the existing SDK APIs.

64-bit DAMs

The DAMs provided with the 64-bit version of Omnis Studio 8.0 use 64-bit architecture. This means that you will need to install separate 64-bit clientware where appropriate. The 64-bit DAMs are not interoperable with 32-bit client libraries and vice-versa. For single-tier and embedded DAMs, including DAMSQLITE, DAMOMSQL, DAMMYSQL, DAMPGSQL and DAMAZON, all necessary changes have been made. The 64-bit ODBC DAM requires the 64-bit ODBC Administrator library and should be used with 64-bit ODBC Drivers to ensure compatibility.

For further details on the Omnis DAMs and clientware configuration, please refer to the Omnis website: www.omnis.net/dams

HFS and Path Separators

The 64-bit version of Omnis Studio on macOS does not support the HFS file system, instead it supports the POSIX file system. Therefore, you need to convert all HFS file paths in your libraries, which are colon-delimited, to POSIX file paths which are delimited with ‘/’ (forward slash). You can convert an HFS path to a POSIX path using the FileOps function $converthfspathtoposixpath().

The sys(9) function can be used to insert the correct path separator on the current platform. On the 64-bit version of Omnis Studio on macOS the function returns ‘/’ (forward slash) as the path separator.

Shared Access to Libraries and Datafiles

The 64-bit macOS version of Omnis Studio does not support shared access to libraries and datafiles. Specifically, the Omnis Datafile Databridge *must be used* to share Omnis datafiles on 64-bit macOS.

Consequently, there is a difference with regards to the $shared property for libraries and datafiles. For libraries ($root.$libs.LIB) on 64-bit macOS, the $shared property *cannot be set* to true as shared access is not supported: in effect, this property is redundant on this platform.

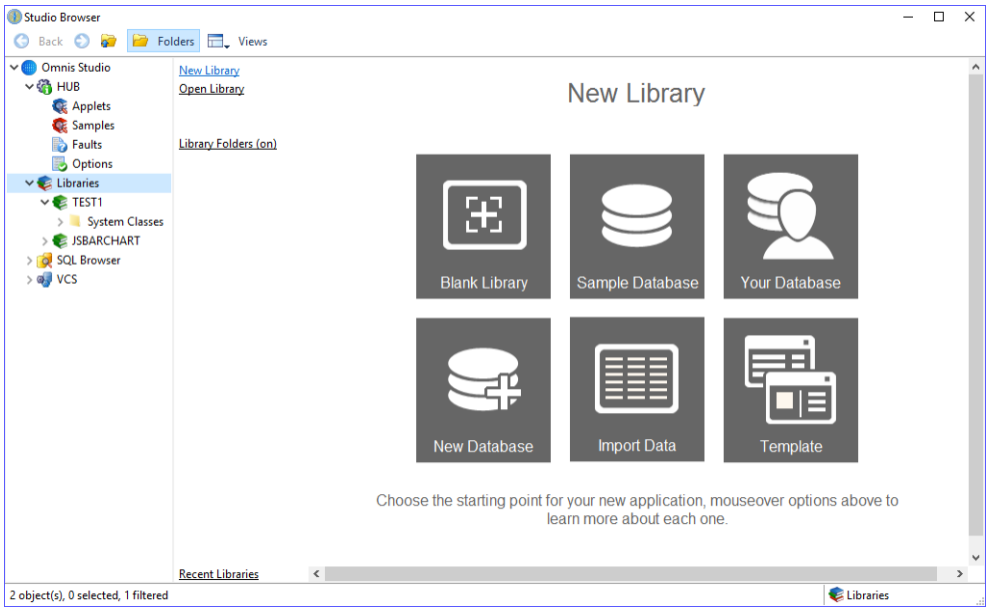
For datafiles ($root.$datas.DATA) on 64-bit macOS, the $shared property cannot be set to true for a datafile opened as writeable unless the Omnis Datafile Databridge is being used.

App Builder

The **App Builder** is a new tool that allows you to create an application from a number of different options including a sample database, your own database, or by importing some data, for example, from a spreadsheet. From the initial selection, the App Builder lets you set up the tables (including primary keys), select the layout for your web and mobile forms, then you can select the color themes, branding, and the navigation in your app, and finally Omnis will build the application for you which you can test straight away in a web browser.

Creating a New Library

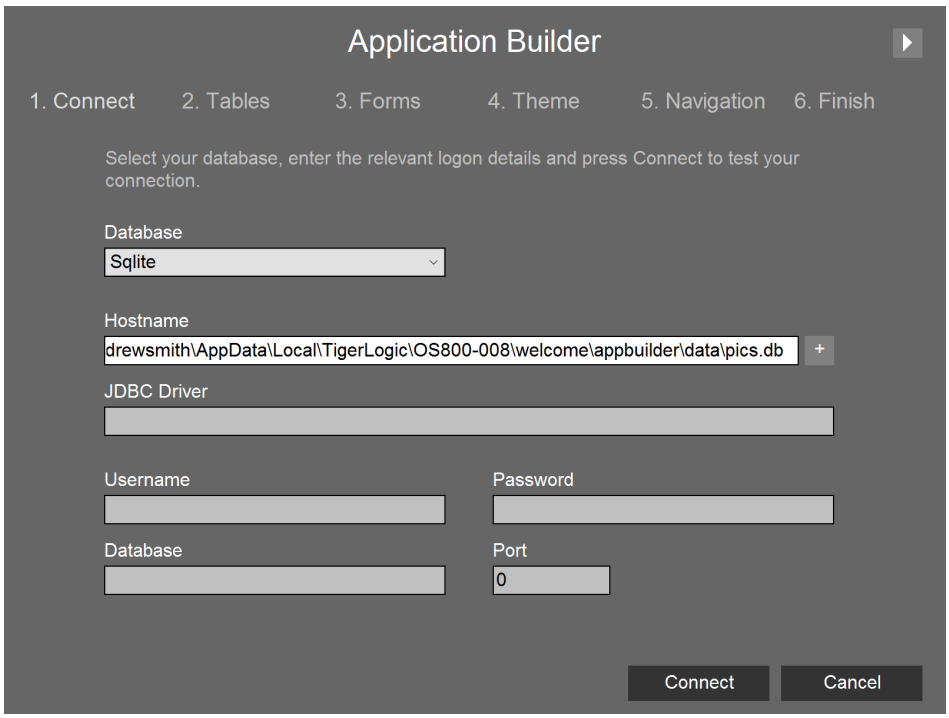
The App Builder is available when you choose the **New Library** option in the Studio Browser, and will allow new users to create apps quickly or evaluate Omnis Studio, and it will allow existing users to prototype apps and web forms quickly based on an existing database or from a template.



When you have created your application you can test it in your web browser. The library you have created is loaded into the Studio Browser under the **Libraries** option. You can examine the classes in the library and start to make modifications or add further classes.

Creating an app from your Database

The App Builder lets you create a new app (library) based on an existing database. To do this, click on Libraries in the Studio Browser, click on New Library, select the 'Your Database' option and step through the process of building a library. For this option you'll need to be able to access and logon to your database: for most types of database you'll need the hostname, username and password, and for some other database types you might need additional information.



After logging on to your database and connecting, you then need to select the tables you want to include in your app, select the type and layout for the web forms (JavaScript remote forms), then you can select a color theme and add your company logo, then choose the navigation scheme (either a list or menu), and finally you can test the app in your web browser or open the library in the Studio IDE.

Developer Hub

The Developer **Hub** is a new section in the Studio Browser that provides useful information for developers, such as the status of the most recent reported and fixed faults, together with information, videos and tips for new Omnis developers. For new users the Hub is the default option, but you can change this and the contents of the Studio Browser under the **Options** setting.

Hub

The Hub option itself contains information and videos for new and existing developers, including ‘How to’ videos about useful functions in Omnis Studio.

Applets and Samples

The **Applets** option provides a number of example Omnis applications that show the full capabilities of Omnis Studio for building web and mobile applications. You can open each of the examples in a web browser (when you select an example it is opened in your browser automatically), and you can examine the code in the associated library under the **Libraries** option in the Studio Browser.

The **Samples** option provides a number of sample Omnis libraries demonstrating specific components or programming techniques in Omnis. Once you have opened the sample library, you can examine its classes and underlying code under the **Libraries** option in the Studio Browser.

You can use the Omnis libraries under the Applets and Samples option as templates for your own libraries, or you can reuse individual classes or the Omnis code within the libraries.

Faults

The **Faults** option provides information about the latest *Reported* and *Fixed* faults in Omnis Studio – this is real-time information so you can check the most recent faults. If you have reported a fault in Omnis Studio you can check its status here.

Options

The **Options** setting allows you to configure the behavior, contents and appearance of the Studio Browser.

Show These Tools

Under the 'Show These Tools' option you can specify which tools (nodes) are displayed in the Studio Browser. By default, the **SQL Browser** and **VCS** are shown, but the **Omnis datafiles** browser is not displayed. If you wish to open and browse Omnis datafiles you will need to enable this option.

Default Browser Node

The 'Default Browser Node' option lets you specify whether the Hub or Libraries node is displayed by default when Omnis starts up.

New Libraries

The 'New Libraries' option lets you enable or disable the App Builder which will be opened when the 'New Library' option is selected in the Studio Browser: see elsewhere in this manual for information about the App Builder. When the App Builder is disabled the New Library option will prompt you to create a new blank library (as in previous versions) with no data classes, SQL login setup, or remote forms.

Appearance

Appearance Theme

The Appearance option allows you to change the theme used in the Omnis Studio IDE. Together with the Default theme which is loaded by default, there are a number of other themes from which you can choose. See the "Color Themes and Appearance" section later in this manual.

Window Frame Theme

(Windows only) The 'Window Frame Theme' option allows you to set the color theme or style for the frame edge of windows and forms. The options are Default, Windows 7, 8, or 10.

Code Assistant

A number of enhancements have been added to the Method Editor to allow you to write Omnis code faster and more efficiently, including the Code Assistant and the Method History list which is described at the end of this section.

The **Code Assistant** is a new tool that will help you write lines of code in an Omnis method, and therefore will allow you to create apps more easily in Omnis Studio. The new code assistant will pop up *automatically* at the insertion point when you type a command parameter or some notation in the *method editor,* or you can open it at the appropriate place in the method editor using **Ctrl-Space.**

The Code Assistant only opens *when the caret is visible* in the method editor, i.e. it can only open when no text is selected in a line of code. Specifically, it will pop up when the caret is positioned at *the end of some text* which is either at the end of the entry field content in the method editor, or prior to some type of delimiter in the expression syntax, e.g. a function separator character. The automatic popup is delayed by a timer which is specified in a new Omnis preference called $codeassistanttimer (in Omnis.$prefs).

Further highlights of the new Code Assistant include:

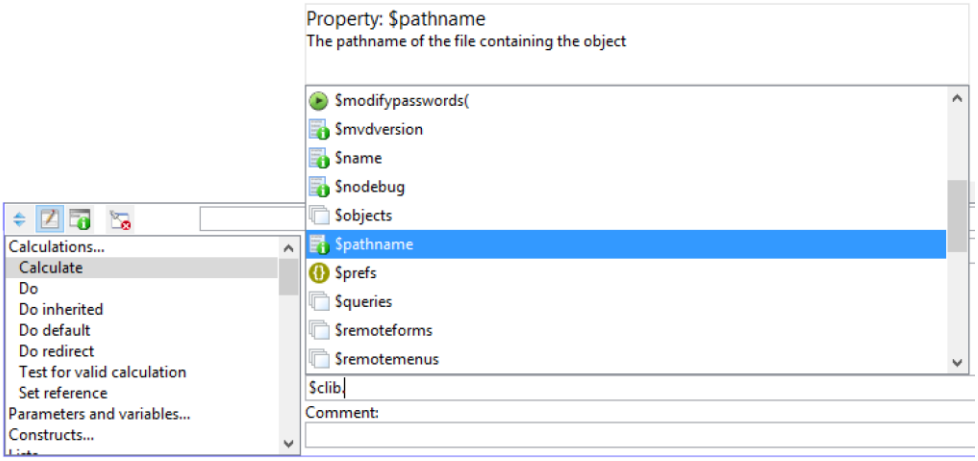
- In addition to Ctrl-Space, you can use various special keys to navigate the popped list and request further help.
- It provides assistance entering notation relative to an item reference and functions.
- It displays method descriptions, method definitions and parameter descriptions.
- Assistance entering notation relative to a group method, as well as notation relative to $ref in the parameters of a group method.
- Intelligent generation of the list of possible values to assign to a property.
- Property and method tooltips in the method list.
- Assistance for initial values and when using expand entry-box in the method editor.
- An improved expand entry-box interface.
- Replaces existing data when selecting an item in the Code Assistant popup.
- Assistance entering certain commands such as *Do method.*
- Parameter highlighting, including parameters for commands such as SMTPSend.
- Parenthesis and square bracket matching.
- Assistance entering methods with overloaded definitions.

Note to existing developers: The Code Assistant is a replacement for the Notation Helper, present in previous versions of Omnis Studio, which has many advantages and improvements over the previous notation helper. The new Code Assistant pops up sooner than the old Notation Helper, so we hope you will find it quicker and easier to use.

Short Cut Keys and Help

You can manually request the Code Assistant popup to open by typing **Ctrl-Space** – this will work on Windows PCs and Mac keyboards. The Ctrl-Space shortcut key will only work if some code assistance is available for the syntax item *to the left of the current insertion point.* This short cut key is a de-facto standard used to request code assistance in many other development tools so should be familiar to developers.

The Code Assistant supports the **Page up, Page down, Home** and **End** keys, to navigate the popped up list. When you use these keys, or **Up Arrow** or **Down Arrow,** the Code Assistant displays help information about the currently selected line in a help panel above the popped list, for example, the following image shows the help text for $pathname which is a property of the current library ($clib).



Short Cut Key Summary

Key	Action
Ctrl-Space	Opens the Code Assistant
Page up, Page down	Displays next or previous 'page' in the popup list or Help pane
Home and End keys	Moves to the beginning or end of the popup list
Up or Down Arrow	Moves up or down the popup list
Return or Enter	Select the current line in the popup list

What Help does the Code Assistant Provide?

In most cases the Code Assistant will popup automatically at the cursor if it can provide help for the current item in your code or context, however the following sections detail the behavior and function of the assistant with regards to different items or contexts in which Omnis provides you with help.

Item References and Notation

In order to provide code assistance, Omnis needs to be able to look up a notation string and map it to the table of methods and properties that apply to the current addressed item. In order to do this for notation paths that start with an item reference, Omnis needs a new piece of information that identifies the notation you intend to use with the item reference – this item is called the *item reference class* and the method editor allows you to select an item reference class as the subtype of an item reference. The class works in the same way as the subtype of an object reference, meaning that the item reference class is solely used to provide code assistance – no check is ever made to see if the item reference is being used at runtime to address items that correspond to its class.

Item reference classes use a similar hierarchical scheme to notation paths. Example classes are $iwindows.window, and $iwindows.window.$objs. There are some special classes that include * in their path. For example,

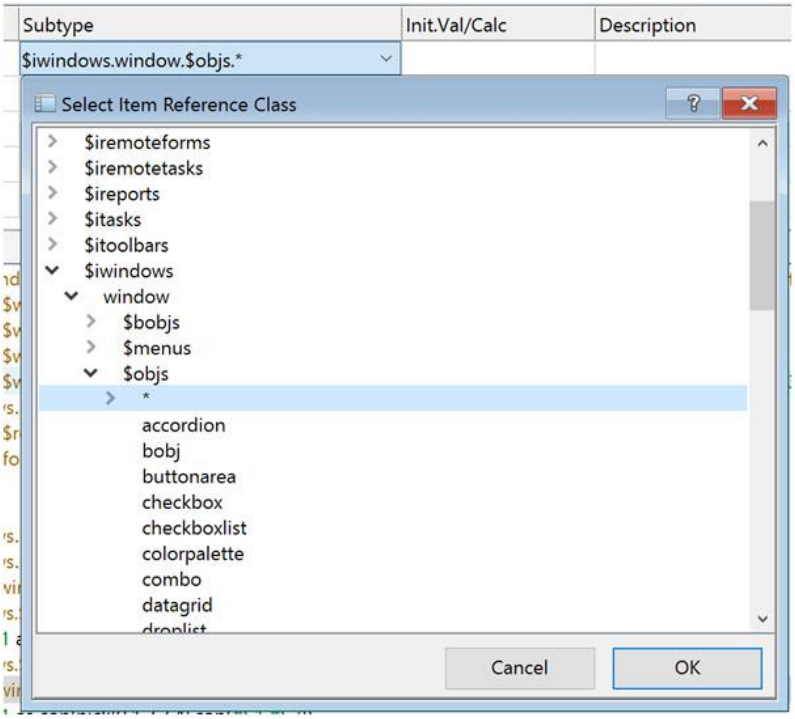
```
$iwindows.window.$objs.*
```

accumulates all possible properties and methods for the possible children of $objs (there is a child for each object type), and is used when the Code Assistant cannot isolate the class of the member of $objs to a single object type.

$iwindows.window.$objs.*.$objs accumulates all $add, $addafter and $addbefore methods for all containers, and is used when the Code Assistant cannot determine the type of a container.

Code assistance for notation works as follows:

- The Code Assistant takes the notation path (and the item reference class if necessary and available) and looks up the matching item reference class.
- If it cannot determine a class, then the Code Assistant provides no assistance.
- If the Code Assistant can determine a class, then it pops up the methods and properties that match the currently entered prefix.



Functions

Code assistance is available for functions, and static methods implemented by external components. The latter is provided by a two step process, where you first select the component from a popup, such as FileOps.$, and you then select the static method from an automatically popped up list of static methods.

Method Information

The Code Assistant displays method descriptions and parameter information in the help panel when a method is selected in the popup. This information is available for all types of method, including functions, external component methods, built-in Omnis notation methods, and your own custom methods.

Group Methods

Methods such as $add and $findname for a notation group return an item reference to a member of the group (assuming they work). The Code Assistant assumes that the call will work, and provides assistance for notation entered after the group method, e.g. if you enter $cinst.$objs.$add(kEntry,0,0,100,100), then as soon as you enter a dot (.) after this expression, you get assistance for all objects that could be in the group (Omnis does not parse the $add call and attempt to provide help for the specific object type added).

$ref

When you use group methods such as $sendall or $makelist, you use $ref in the parameters of the group method to refer to a member of the group. The Code Assistant provides help for $ref, by using the relevant item reference class for the group member (provided it can identify the item reference class of the group).

$assign

When you enter . (dot) after a property name, the Code Assistant provides $assign and $canassign as possible options. If you select $assign, you will be prompted with a popup that provides either all initial items you can enter, or the list of constants or strings which make sense to assign to the property. The latter always applies when the Code Assistant can determine the list of constants or strings which make sense.

In addition, when you are coding a Calculate statement, if you enter a path to a property in the field name field in the method editor, then when you move to the

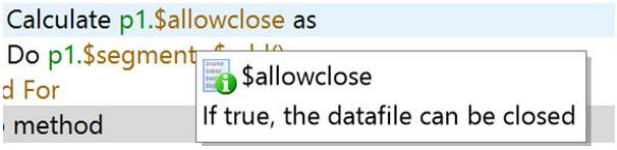
calculation field, provided that the calculate field is empty, the Code Assistant will popup the list of constants or strings which make sense to assign to the property.

For example, enter $cwind.$objs.test.$backcolor as the field name, and move to the empty calculation field. The popup will contain a list of color constants.

If you wish to assign something else, start typing that, and assistance will revert to normal. The only restriction here is that if you type k (when the values that make sense are a list of constants), you will only see the constants that make sense, rather than all constants.

Tooltips

The method editor displays a tooltip when you position the pointer over a property of a method name in the listing of the method. This shows you the property description, or the method interface and description. The tooltip for a constant also shows you the constant description.



Initial Values

You can use the Code Assistant in initial value column of the variable pane of the method editor.

Expanded Entry

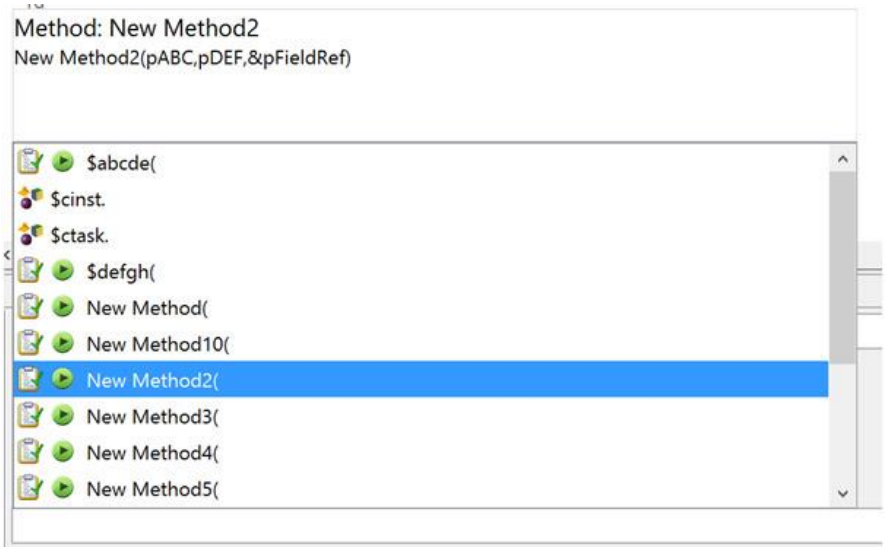
The Code Assistant is available in the expanded entry box in the method editor – it opens as an overlay over the method editor command palette. You can close it by clicking on another window, pressing return (or pressing escape to discard changes).

Replacing Data

When you select some notation from the Code Assistant popup, it replaces the entire word (if any) in which the caret is located.

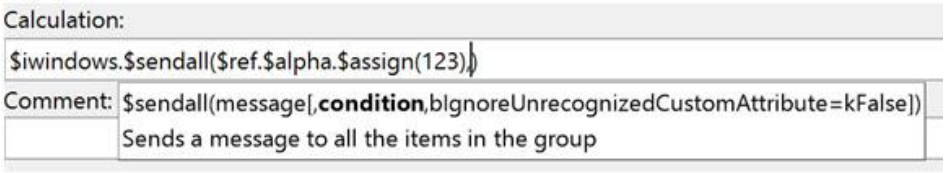
Method Commands

The commands *Do method, Do async method, Do code method, Load error handler, Unload error handler, Set 'About' method, Set timer method, Start server, Install menu, Install toolbar, Open window,* and *Set report name* use a Code Assistant popup to select their method or class.



Parameter Highlighting

When you position the caret somewhere in the parameters of a function or method that the Code Assistant recognizes, or in a method command that has parenthesized parameters e.g. SMTPSend, Omnis displays a popup (in the opposite direction to the Code Assistant popup) that displays the method parameters and the method description. In addition, the parameter in which the caret is currently located is highlighted in bold.



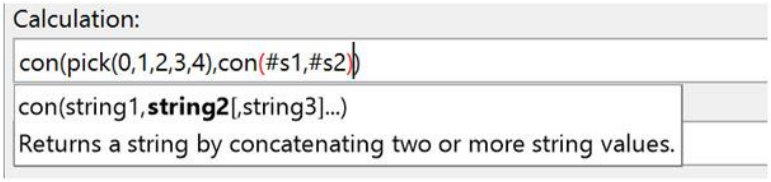
You can press Escape to temporarily dismiss this popup. If you want to disable it, you can change an entry in the config.json configuration file, by setting the “parameterHelpEnabled” boolean property in the “ide” group to false.

In addition, the configuration allows you to specify the maximum width of the parameter help popup (it defaults to half the screen width) in “parameterHelpWidth”.

You can also specify functions or methods you wish to exclude from parameter help in the “parameterHelpExclusions”. Possible examples of what you might want to exclude are “con” and “$assign”.

Parenthesis Matching

When you position the caret immediately after an open or close parenthesis in an expression, or immediately after an open or close square bracket, Omnis draws the matching parentheses or brackets using a different color.



There are two properties which control this, in the method editor chroma coding options: $bracketbackcolor and $brackettextcolor. To disable this, you can set both of these options to kColorDefault.

Overloads

Certain methods are overloaded. In simple cases, the Code Assistant shows this by using a vertical bar to separate different possibilities e.g.

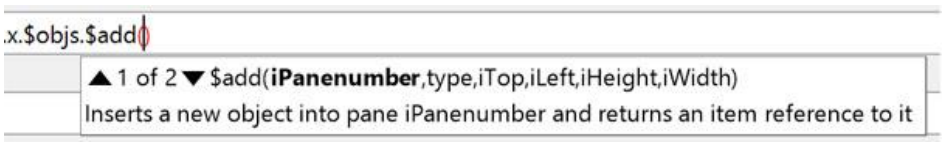
```
$remove(rLine|iLineNumber|kListDeleteSelected|kListKeepSelected)
```

However, there are other cases where this is not possible, for example:

```
$createobject for a JavaObjs\System\java\lang\String object has 15 overloads
```

$add for an unknown window object could be adding a complex grid or paged pane or scroll box, and the object being added may or may not be an external component.

In these cases, the description shown for the method shows all overloads, and the parameter highlighting popup has arrow icons, indicating that you can use the up and down arrow keys to select the overload you are using, thereby resulting in sensible parameter highlighting. Omnis does not attempt to figure out the matching overload by analysing the parameters.



Method History

Omnis now stores a list of visited methods which allows you to quickly move back to a recently visited method. The toolbar in the Method Editor contains a **Back** and **Forward** button allowing you to traverse the history of visited methods. Note that inherited methods and the object nodes in the method editor do not form part of the history which can hold up to 256 items. You can also use F10/Shift-F10 to move back and forward respectively. In addition, a long press on either of the buttons opens a menu which shows the available history items in the direction of the button, up to a limit of 20 menu items.

Omnis removes affected entries from the history when a library is closed, a class or method is deleted, or when various other actions occur that would affect an entry in the history list, such as when fields are renumbered.

Command Blocks

The start and end of any block commands are highlighted in the Method Editor. This includes the commands If, While, For, Repeat, Switch, and Begin which highlight the associated closing commands (e.g. else, until etc) when one of the statements that makes up the construct formed by the commands is selected in the method editor. For example, if a **For** statement is the current line, then the "End for" and "For" will both be highlighted. Or if a **Case** statement is the selected line, then all cases in the same switch, "Default", "Switch" and "End switch" will all be highlighted. The style or color of the highlighting uses a pair of new chroma coding options, $currentblocktextcolor and $currentblockstyle.

Client-side Scripting

When you add a new method to a remote form, and the method editor recognizes the method name as a known method that should be client-executed, then the method editor now marks it as client-executed, and also populates its parameters (if any). This applies to new methods containing the following methods: $candrop, $drag, $filereadcomplete, $init, $term, $getscrolltip, $filtergrid, $sortgrid, $sfsorder, $sfscanclose, $pushed, and $sqldone.

In addition, from 8.0.2 onwards, Omnis adds boilerplate code to the $sqldone method: see earlier in this manual for details.

Method Notes

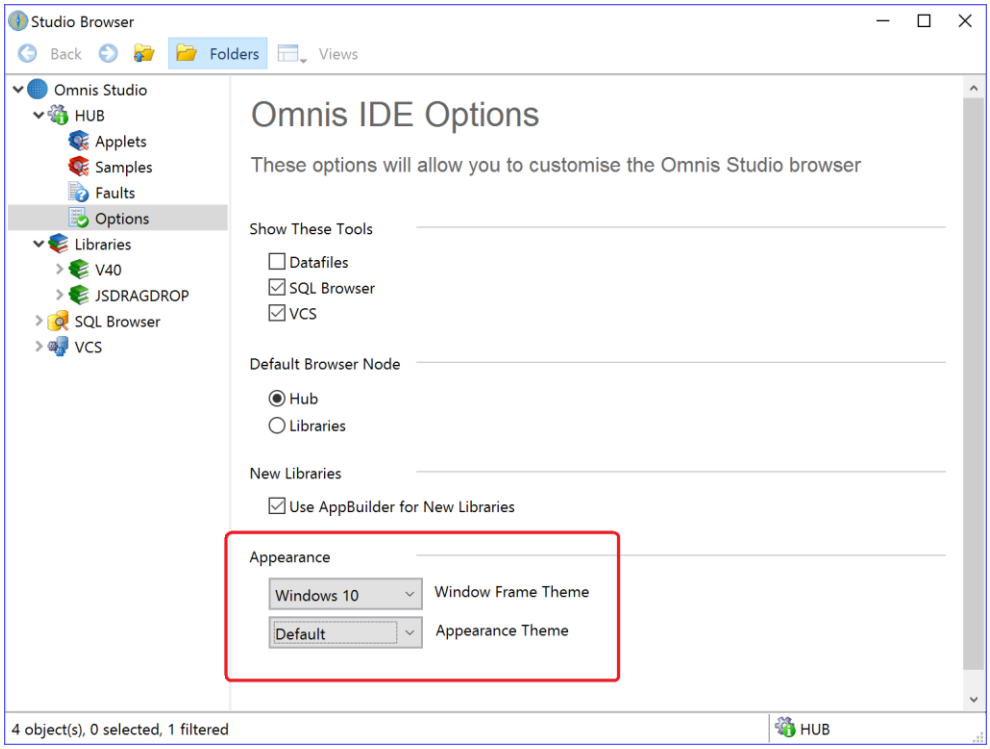
You can now add notes to a method to allow you to document each method in a class. The notes are stored in the $notes property for the method. The notes can be edited on the notes tab in the method editor.

Existing users should note that the $notes property is the $httpnotes property renamed to $notes, which is now available for all methods in a class.

Color Themes and Appearance

It is now possible to change many of the color theme used in the Omnis Studio IDE, and by using the appropriate configuration files, you can manage the color theme used in your deployed apps. Note the following implementation is available on Windows and the new 64-bit, Cocoa based version of Omnis Studio on macOS only, but not for Linux.

The colors used throughout the Omnis Studio IDE are now stored as a set of properties or *theme* which can be changed under the **Options** setting in the Hub in the Studio Browser. There are a number of themes to choose from, including the 'Default' theme which is intended to match the colors used on different platforms supported in Omnis as closely as possible. You can change individual colors or settings within a theme, and in this case, your modified theme will be saved as a 'Custom' theme alongside those provided.

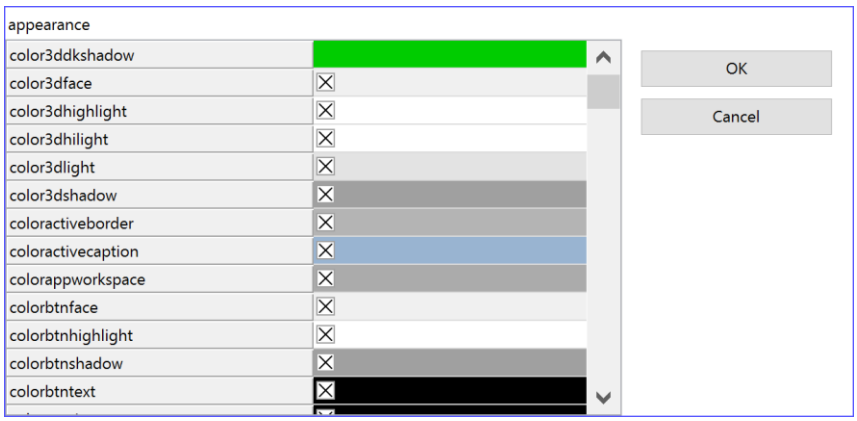


For Windows only, you can also change the theme used for the outside edge or frame of windows used in the Studio IDE and your own windows, which can be set to match the style of the window frame displayed on various Windows platforms.

Appearance Property

The current theme in the Omnis Studio IDE is stored in a new Omnis preference called $appearance, which can be edited in the Property Manager. To edit this property, click on 'Omnis Studio' in the Studio Browser, click on Prefs, then select the Appearance tab in the Property Manager and click on the droplist next to the $appearance property.

Note that when editing $appearance in the Property Manager, the default colors may not always draw correctly since the editor grid itself uses the configured colors, and therefore if a color is set to the default setting, there is a check mark icon (an X) to the left of the color.



In addition, there is a new Omnis preference called $windowoptions that stores the current Window Frame theme which you can also edit in the Property Manager: this is described at the end of this section.

Appearance and Theme Files

The color and appearance settings used in the Omnis Studio IDE, and displayed in $appearance in the Property Manager, are stored in a new configuration file called **'appearance.json',** which is located in the Studio folder in the main Omnis folder. This file contains the current color theme settings in $appearance which will either be the default theme, or one of the other themes provided, or a custom theme.

The Default and other themes (Blue, Green, etc) are stored in a separate JSON file in the 'studio/themes' folder. As you change the theme setting under the Options setting in the Hub, the appropriate JSON theme file is written to the appearance.json file in the Studio folder.

When Omnis loads, it will copy the appropriate theme file into /studio/appearance.json. If you alter one of the theme colours using $appearance in the Property Manager, Omnis will recognize that the default or one of the built-in themes has changed and therefore will create a custom theme 'appthemecustom.json' in the themes folder. In this case, when Omnis restarts, it will load the Custom theme and it will add it to the droplist in the Options setting in the Hub.

When a theme has been changed, if you then try to switch to another theme using the Hub, you will receive a Yes/No message asking you to confirm if you wish to overwrite the custom changes.

Appearance Configuration File Contents

As well as choosing a theme in the Options setting in the Hub and chaging colors and settings via $appearance in the Property Manager, you may like to edit the configuration files – we recommend that you do not change the default theme file in the studio/themes folder, but you can make a copy of it and edit that. The color number values in the configuration files are 32 bit integers: if the value is less than zero, it is a representation of a colour constant in the form 0x8nnnnnnn where nnnnnnn identifies the constant, otherwise, the value is RGB - 8 hex digits: 00bbggrr e.g. 0x0000ff00 is full green.

The following settings are available in the appearance.json file which correspond to the settings under the $appearance property in the Property Manager.

Tab Panes

colortab, colortabhot, colortabselected, colortabdisabled, colortabborder
Colors used for tab panes with $tabstyle set to kDefaultPanes. If these are all set to kColorDefault, then Omnis uses the system theme; otherwise, Omnis imitates the system theme and draws its imitated tab pane using these colors.

iconidtabpaneleftbutton, iconidtabpanerightbutton
The icon ids of the arrows drawn on tab pane scroll buttons.

Lists

colorlistlineselectedwin, colorlistlinehotselectedwin, colorlistlinehotwin, colorlistlineunfocusedwin
These colors are used for Windows only. Used for the background of “lines” in different states, e.g. lines in headed list boxes, tree nodes etc. If these are all set to kColorDefault, then Omnis uses the system theme; otherwise, Omnis imitates the system theme and uses these colors.

colorlistevenrowbackground
Color used to highlight even rows in lists, on macOS only; it has no effect on Windows.

Headed Lists

colorheader, colorheaderhot, colorheadpressed, colorheaderseparator
Used in the header of headed list boxes. If these are all set to kColorDefault, then Omnis uses the system theme; otherwise, Omnis imitates the system theme and uses these colors.

Buttons

colorpush, colorpushborder, colorpushhot, colorpushhotborder, colorpushpressed, colorpushpressedborder, colorpushpressedtextmacos, colorpushdefault, colorpushdefaultborder, colorpushdefaulttextmacos, colorpushdisabled, colorpushdisabledborder
Used for system push buttons and various other controls that use the same theme. If these are all set to kColorDefault then Omnis uses the system theme; otherwise, Omnis imitates the system theme and uses these colors. The …macos properties are used for text on the button in the relevant state, on macOS only.

colorpushdefaultflash
A Boolean which indicates if the default button is to flash (or pulse).

Radio buttons and Check boxes

checkradiobordermacos, checkradiounchecked, checkradiochecked, checkradiouncheckedpressed, checkradiocheckedpressed, checkradiomark, checkradiomarkhot, checkradiomarkdisabled
Used for system checkboxes and radio buttons. If these are all set to kColorDefault Omnis uses the system theme; otherwise, Omnis imitates the system theme and uses these colors. The checkradiobordermacos property is only used on macOS, for an unchecked control: on Windows, the theme uses the mark color as the border color.

Tree Lists

colortreeiconcollapsed, colortreeiconexpanded, colortreeiconhot
Used for the standard arrows in tree controls. If these are all set to kColorDefault Omnis uses the system theme; otherwise, Omnis imitates the system theme and uses these colors.

Borders

colorctrleditborder, colorctrleditborderinsetmacos, colorctrllistborder, colorgroupboxborder
Used for the borders of controls that use a relevant kBorderCtrl… value. If the color value is kColorDefault, Omnis uses the system theme. Otherwise, Omnis imitates the system theme and draws the border using the specified color.
colorctrleditborderinsetmacos is only used on macOS on retina displays - for edit controls, the border is 2 single pixel rectangles, one inside the other and this color is for the inset rectangle.

Group Boxes

colorctrlgroupboxmacos
the background color of a group box on macOS, when the border of the group box is kBorderCtrlGroupBox.

Reports

colorreportdesignposnsectiontext
the color of the text on a report design mode section, for a positioning section.

Page and Print Preview

colorpreviewpaper
the color of the paper in the page preview window. kColorDefault results in white; otherwise Omnis uses the specified color.

colorpreviewfoundhighlight
the color of the window that briefly appears to indicate the location of text found when searching a print preview. kColorDefault results in red; otherwise Omnis uses the specified color.

colorpreviewfoundtextbackground
the color of the background on which found text is drawn, when searching a print preview. kColorDefault results in yellow; otherwise Omnis uses the specified color.

colorpreviewfoundtext
the color in which found text is drawn, when searching a print preview. kColorDefault results in red; otherwise Omnis uses the specified color.

Scrollbars

The colors used for scrollbars are now configurable in the Property Manager under the $appearance preference (and stored in appearance.json). The configured colors apply to all scrollbars except those on system dialogs and third party controls (e.g. OBrowser). If all colors relevant to the platform are default, then the standard system scrollbar is used.

colorscrollbar
The color used to fill the scrollbar.

colorscrollbararrowwin
Windows only. The color of a scrollbar arrow on an enabled scrollbar when the mouse is not over the arrow button, and when the button is not being pressed.

colorscrollbardisabledarrowwin
Windows only. The color of a scrollbar arrow on a disabled scrollbar.

colorscrollbarhotarrowwin
Windows only. The color of a scrollbar arrow on an enabled scrollbar when the mouse is over the arrow button, or when the button is being pressed and the mouse is not over the button.

colorscrollbarhotbuttonwin
Windows only. The color of a scrollbar button on an enabled scrollbar when the mouse is over the button, or when the button is being pressed and the mouse is not over the button.

colorscrollbarhotthumb
The color of the scrollbar thumb when the mouse is over the scrollbar (macOS) or over the thumb (Windows).

colorscrollbarthumb
The color of the scrollbar thumb when the mouse is not over the scrollbar.

colorscrollbartrackingarrowwin
Windows only. The color of a scrollbar arrow on an enabled scrollbar when the button is being pressed and the mouse is over the button.

colorscrollbartrackingbuttonwin
Windows only. The color of a scrollbar button on an enabled scrollbar when the button is being pressed and the mouse is over the button.

colorscrollbartrackingthumbwin
Windows only. The color of the scrollbar thumb when the thumb is being dragged.

colorscrollbarwarmthumbwin
Windows only. The color of the scrollbar thumb when the mouse is over the scrollbar but not the thumb.

Menus and Window Menu bars

Checked menu lines on the Windows platform draw the checked indicator based on colorlistlineselectedwin unless colormenu is set to kColorDefault (in which case they draw using the system default). The color of the checkmark (when colormenu is not set to kColorDefault) is either colormenutext or colorgraytext (the latter for disabled menu items).

The hierarchical menu indicator now draws using colormenutext.

The following colors apply to Window Menu bars.

colormenuwindowmenubarwin
The color of a window menu bar on the Windows platform. If set to kColorDefault, the menu bar draws using the system theme.

colormenuwindowmenubartextwin
The color of text drawn on a window menu bar on on the Windows platform. If set to kColorDefault, the menu bar draws using colormenutext.

colormenupenwin
The color of separator lines, and the vertical gutter line, drawn on the menu (rather than the bar) on the Windows platform. If set to kColorDefault, the lines draw using color3dlight.

Toolbars and Docking Area

colortoolbarobjecthighlightwin
Windows only. The color used to draw the rectangle that highlights toolbar items. kColorDefault means Omnis will use the system theme. Otherwise Omnis imitates the system theme using this color.

dockingareacolor and **dockingareatextcolor**
The color used to draw the background area of the main Omnis toolbar docking area, and the text.

Client Methods

clientexeccolor, inheritedcolor, nosetpropertycolor, runtimepropertycolor, setpropertycolor, toolobjselectcolor
These replace the notation in $prefs previously used to set these values. If set to kColorDefault, Omnis uses the color to which the property defaulted.

Chroma Coding

bracketcolor, bracketbackcolor, commentcolor, ctrlkeywordcolor, currentblockcolor, keywordcolor, stringcolor, variablecolor
These replace the chroma coding options for the method editor (the option to edit these has been removed from the method editor modify menu). If set to kColorDefault, Omnis uses the color to which the property defaulted.

commentstyle, ctrlkeywordstyle, currentblockstyle, keywordstyle, stringstyle, variablestyle
These replace the chroma coding options for the method editor. These are integer values representing a style (see the values of the style constants which can be ORed together, e.g. kBold has the value 1).

Highlight Color

colorthemered
A color used where Omnis needs to make something stand out. kColorDefault maps to red.

This is used in the following places: #STYLES editor, and the Report class editor when marking the highlighted section Watch panel in method editor, to make a changed variable value Data grid ($userdefined true) to mark the current selected column.

In addition, the drag bitmap when dragging more than one line on macOS shows the count using a circle drawn in highlight color, with text in highlight text color.

Property Manager

colorpropertymanager
The color of the property manager background. If left as kColorDefault, Omnis defaults to 3dface on macOS, and the toolbar background color for Windows.

Changing and Testing Colors

For testing purposes, you can add two files to the Studio folder, which are copies of appearance.json: they are 'appearance_custom.json' and 'appearance_default.json'. These files must have the same syntax as appearance.json. The default file should contain all colors set to kColorDefault (the initial state of appearance.json). The custom file can contain any color scheme you like. You can then use F11 to select the custom scheme, and shift-F11 to select the default scheme, while running Omnis. This behavior can be disabled by setting the config.json entry "allowSwitchAppearance" in the "ide" object to false, or alternatively by not including these additional files.

Additional Notes

System dialogs and Menus

System dialogs (file dialogs etc) do not use the theme colors.

Menus on macOS do not use the theme colors. On Windows, the colorlistlineselectedwin color is used to highlight menu lines.

JavaScript Client

The JavaScript client will use the system colors configured on the Omnis App Server running your app. Therefore if you want to use a specific theme for your deployed web and mobile apps, you need to copy your appearance.json file to the Omnis App Server.

macOS and Cocoa

The term system theme is a little loose for Cocoa. Omnis tries to match the system theme, but unlike Windows, there are not always APIs in the OS to perform the drawing (hence some system theme drawing on Cocoa is actually Omnis imitating the OS theme).

Some of the theme background colors have been rationalized (this does not apply to Carbon) as follows:

kBGThemeWindow, kBGThemeContainer and kBGThemeTabStrip now fill with kColorWindow on both platforms. Previously these used kColor3DFace on Windows.

kBGThemeTabPane now fills with the selected tab background (using either the system theme or colortabselected - the system theme applies with colortabselected is kColorDefault).

Window Frame Appearance on Windows

In addition to the color management outlined above, you can change the appearance of the frame edge of window classes (on Windows operating systems only), which allows you to comply with the latest style for window frame edges on Windows 8, 8.1 and 10. For most purposes you can accept the default settings for the current Windows platform, but you can change the frame theme if you want.

You can change window frame themes under the **Options** setting under the **Hub** section of the Studio Browser (Windows only). You can select Default, Windows 7, 8, or 10 which allows you to view how your application will appear on different Windows platforms.

In addition, there is a new property in the Omnis preferences, $windowoptions, which allows you to edit the appearance of window frames in libraries running on Windows – note this preference is only editable on the Windows platform.

Window Frame Configuration files

There is a new file called 'window.json' in the Studio folder, which stores the values of the $windowoptions preference. The window.json file configures the appearance of the window frame, and also configures the operating systems for which the configured appearance will be used, including the old appearance for Windows 7, and the new configured appearance for later Windows operating systems. There are a number of theme files in the 'studio/themes' folder which are copied to window.json as appropriate.

Active Caption Colors

useborderactivecolorfordefaultactivecaption
specifies the color of the active caption in Windows (stored in window.json). This is an integer with 3 possible values.

If it is zero, the behavior is as before (the active caption defaults to white if it is set tokColorDefault).

It it is one, and the active caption color is kColorDefault, then the active caption color is the same as the active border color.

If it is two (the default), then the default active caption color depends on the system setting on the accent color settings panel: "show colour on start, taskbar, action centre and title bar" - if the system setting is off, then this is equivalent to useborderactivecolorfordefaultactivecaption equal to zero - if the system setting is on, then this is equivalent to useborderactivecolorfordefaultactivecaption equal to one.

Note that this applies to both small and normal size captions, and only applies when the relevant caption colour is kColorDefault.

Drag and Drop Data

Drag and drop for the JavaScript Client provides the ability for end users to drag data from one JavaScript control in a remote form, and drop that data onto another JavaScript control. In addition, end users can drag files from their desktop and drop them onto a JavaScript control within a remote form displayed in their web browser.

IMPORTANT NOTE: Support for drag and drop in JavaScript remote forms is limited to desktop browsers only, including Chrome, Edge, Firefox, IE 11, and Safari – drag and drop is *not supported* in mobile browsers.

To drag and drop some data, the end user can click and hold down the pointer over a JavaScript control on a remote form, then drag the highlighted control onto another control and release the pointer. To enable drag and drop, you have to set various properties in the source and target JavaScript controls, and handle various events in each control as the drag and drop events occur.

Existing users should note that the event constants and their parameters work in a very similar manner to those for the drag and drop mechanism in the thick client, with the addition of a new constant pDropId which identifies the area of a control over which the drop is to occur (see under Events).

Example Library

There is an example library demonstrating how you can drag and drop images between JavaScript controls, and the library allows image files to be dropped onto a control from

the desktop. The example library is available in the JavaScript Component Gallery which is on the Omnis website: www.omnis.net

Dragging Data

Dragging data is limited to certain data-bound JavaScript controls and is not possible for all types of JavaScript controls. JavaScript client controls that support dragging data will have the **$dragmode** property. This can be set to either **kNoDragging** or **kDragData.**

Note that the $dragiconid property used in the thick client is not supported for drag and drop in the JavaScript client, for a number of technical limitations in various browsers. The dragged image is typically an image of the dragged element created by the browser, using the content of the element when the drag starts – the client performs various temporary adjustments to the element to make the dragged image correspond to the dragged data as appropriate.

Dropping Data

A drop can occur on any JavaScript control, but remotes forms do not accept drops. You can specify that a control can accept dropped data by setting its $dropmode property. When a control can accept some data, the JavaScript client highlights the destination control. For JavaScript client controls, $dropmode can be one of the following constants:

- **kAcceptControl**
  Data from a JavaScript client control can be dropped onto this control.
- **kAcceptFiles**
  Files dragged from the system (desktop) can be dropped onto this control.

In addition, the list, tree and data grid controls have the $hiliteline property, indicating that data can be dropped on a specific list line or tree node rather than the entire control. This also means that rather than highlighting the entire control, the client highlights the current destination line or node when a drop can occur.

Scrolling

When the end user is dragging data, they can scroll a destination control vertically by placing and holding the pointer near the bottom or top of the control. This is useful with long lists, grids or tree controls, when the $hiliteline property is enabled.

Events

In order to process a drag and drop procedure, you have to handle some events in the $event method in the source and target controls. The drag and drop events must be enabled as required in the $events property for a control.

evDrag

The client sends evDrag when the user attempts to start a drag. evDrag must be executed in a client-executed $event method, and it has the following event parameters:

Parameter	Description
pDragType	Always set to the value kDragData
pDragValue	Described in the Drag Values section below.

If you use *Quit event handler* (discard event) during evDrag, you prevent the drag from starting.

Since it is not always convenient to mark $event for a control as client-executed, the client provides an alternative mechanism. You can implement a client-executed method named $drag for the object, with two parameters (type Var): pDragType and pDragValue. $drag returns true if the drag is allowed, false if not.

The client first attempts to call $drag. If $drag exists and returns true or false, then the drag starts or is not allowed to start respectively. If $drag does not exist, or does not return a value, Omnis sends evDrag if it is selected to execute in $events, and if $event is client-executed.

The drag will only fail to start if $drag executed and returned false, or if evDrag was sent and discarded by Quit event handler.

Data grids, lists and tree controls may select a line or node when the drag starts. This will result in a click event being sent just before $drag is called or evDrag is sent. If the click is sent to the server, it will execute in parallel with evDrag or $drag.

evDragFinished

The client sends evDragFinished when the user has finished a drag (released the pointer). It has no event-specific parameters. evDragFinished can be server or client executed.

evCanDrop

The client sends evCanDrop when the pointer is over a control that can accept a drop of the current drag type (kDragData or kDragFiles). evCanDrop must be executed in a client-executed $event method, and it has the following event parameters (note that pDropId is new for Studio 8.0):

Parameter	Description
pDragType	kDragData if data is being dragged from a control, or kDragFiles if a file or files are being dragged from the system
pDragValue	Described in the Drag Values section below. Note that if pDragType is kDragFiles, this is empty during evCanDrop, since information about the files being dragged is not provided by the browser
pDragField	If pDragType is kDragData, this contains the name of the field from which data is being dragged. If pDragType is kDragFiles, this is empty
pDropId	(new for Studio 8.0) The identifier of the area of the control over which the drop is to occur. Either a line number or ident (when $hiliteline is true), or zero if the control is not list-based (or $hiliteline is false).

If you use *Quit event handler* (discard event) during evCanDrop, you prevent a drop on to the current control and pDropId combination.

Since it is not always convenient to mark $event as client-executed, the client provides an alternative mechanism. You can implement a client-executed method named $candrop for the object, with four parameters (type Var): pDragType, pDragValue, pDragField and pDropId. $candrop returns true if the drop is allowed, false if not.

```
If pDragField=$cobj.$name
   Quit method kFalse
End If
```

The client first attempts to call $candrop. If $candrop exists and returns true or false, then the drop is allowed or not allowed respectively. If $candrop does not exist, or does not return a value, Omnis sends evCanDrop if it is selected to execute in $events, and if $event is client-executed.

The drop will only be denied if $candrop executed and returned false, or if evCanDrop was sent and discarded by Quit event handler.

evWillDrop

The client sends evWillDrop when a drop occurs over a control and drop id combination for which which a drop is allowed according to the can drop processing. The client sends evWillDrop to the control being dragged - therefore, evWillDrop is not sent when dragging files from the system. evWillDrop can be server or client executed. It has the following event parameters:

Parameter	Desciption
pDragType	kDragData
pDragValue	Described in the Drag Values section below.
pDropField	The name of the control where the data is being dropped.
pDropId	The identifier of the area of the control over which the drop is occurring. Either a line number or ident (when $hiliteline is true), or zero if the control is not list-based (or $hiliteline is false).

Quit event handler with discard event has no effect on evWillDrop.

evDrop

The client sends evDrop when a drop occurs over a control and drop id combination for which a drop is allowed according to the can drop processing. evDrop can be server or client executed. It has the following event parameters:

Parameter	Description
pDragType	kDragData if data is being dragged from a control, or kDragFiles if a file or files are being dragged from the system
pDragValue	Described in the Drag Values section below.
pDragField	If pDragType is kDragData, this contains the name of the field from which data is being dragged. if pDragType is kDragFiles, this is empty
pDropId	The identifier of the area of the control over which the drop is occurring. Either a line number or ident (when $hiliteline is true), or zero if the control is not list-based (or $hiliteline is false).

Quit event handler with discard event has no effect on evDrop.

The following $event method is behind an image control and processes the dropped data (this is available in the example library).

```
On evDrop
   If pDragType=kDragData
      If pDragField='LeftImage'
         Calculate iLeftImage as iRightImage

         ;  pDragValue is base64, convert to binary for consistancy
         Calculate lBase64 as mid(pDragValue,pos(',',pDragValue)+1)
         Calculate iRightImage as binfrombase64(lBase64)
      End If
   Else
      Calculate lLine as 1
      ; pDragValue can contain many lines use first file only
      Calculate iRightImageIdent as pDragValue.[lLine].4
      Do $cinst.$clientcommand(
         'readfile',row(iRightImageIdent,'iReadFileBin',kTrue))
      ; readfile is a client command - see below
   End If
```

Drag Values

This section describes both the controls for which data can be dragged, and the drag values generated for each drag.

Combo box

pDragValue is the selected text dragged from the current selection in the entry field component of the combo box. To drag text, you must click and hold the pointer somewhere in the selection before dragging.

Data grid

pDragValue is a list. For a single select data grid, the list has one line, containing the list line being dragged. For a multiple select data grid, the list contains the selected lines being dragged.

Entry

pDragValue is the selected text dragged from the current selection in the entry field. To drag text, you must click and hold the pointer somewhere in the selection before dragging.

List

pDragValue is a list containing the list line being dragged.

Picture

pDragValue is a character string containing the URL of the picture being dragged. If the picture is populated using a variable and $mediatype, the URL is a data URL.

Rich text

pDragValue is the selected text dragged from the current selection in the entry field component of the rich text control; note that this is the plain text without any formatting. To drag text, you must click and hold the pointer somewhere in the selection before dragging.

Tree

The tree only supports dragging when it is in dynamic mode (i.e. when $datamode has the value kKSTreeDynamicLoad). pDragValue is a row containing information about the node being dragged. The row has 3 columns:

Column	Description
ident	The ident of the node
tag	The tag of the node (a character string)
text	The node text

If the $hiliteline property is kTrue for a tree control, and the dropmode indicates that the tree is a potential drop target, the client will expand a node when the pointer enters it while dragging.

Tab control

The tab control contains some special logic that allows you to switch tabs while dragging, if this is the functionality you require. For can drop, it sets pDropId to the tab number of the tab under the pointer (or it sets pDropId to the current tab number if the pointer is over an area of the control which is not a tab). To switch tabs, implement a client-executed $candrop method for the tab control which executes:

```
Calculate $cobj.$currenttab as pDropId
Quit method kFalse
```

Dragging and Dropping Files

In addition to dragging and dropping data from one control to another, end users can drag files from their desktop and drop them onto a JavaScript control in a remote form in their browser. There are two new client commands that allow you to process dropped files, using the $clientcommand method.

closefile

The client records file idents (and their JavaScript File objects) in a table. Use closefile to remove the table entry and release resources. You should really do this for every ident passed in the drag value to evDrop, unless you use readfile which removes the table entry after reading the file.

The row passed to the "closefile" $clientcommand has a single column, which is the ident of the file to remove from the table. If you pass a row where the ident is zero, the client removes all entries from the table.

readfile

The readfile client command allows you to read the contents of a file identified by its ident. After attempting to read the file, the client removes the ident from the table, so a call to closefile is not required.

The row passed to readfile has the following structure:

```
row(ident,instance variable name,base64)
```

The columns are as follows:

Column	Description
ident	The ident of the file
instance variable name	The name of an instance variable in the form used to call $clientcommand, that will receive the contents of the file. Note that this is a character string containing the instance variable name, not the instance variable itself
base64	A Boolean. If true, the file is read as base64; otherwise the file is read as text

The JavaScript FileReader which the client uses to read the file operates asynchronously, so a call to readfile starts the file reading process. When the file read is complete, the client calls the client-executed method $filereadcomplete in the form used to call $clientcommand. $filereadcomplete has two parameters:

Parameter	Description
ident	The ident of the file.
error text	Empty if the file was read successfully, meaning that the named instance variable has been populated with the file contents (either as text or base64-encoded text). If not empty, some text describing why the file read failed

Files dragged from system

For file dragging, pDragValue is only populated for evDrop. It is a list of the files dragged from the system, with columns defined as follows:

Column	Description
name	The file name. Note this is just a name, not the path to the file
type	The MIME type of the file if this was determined by the browser before passing it to the drop event
size	The size of the file in bytes
ident	An integer, unique in the context of the client, that identifies this dropped file. You can use this with the new client commands described in the later section about processing files.

Drag and Drop for Thick Client

evCanDrop, evWillDrop and evDrop for the thick client have a new event parameter, pDropId. This is significant when $hiliteline for the control is true, and contains the id of the location in the control where the drop would occur or is occurring, e.g. the list line for a list.

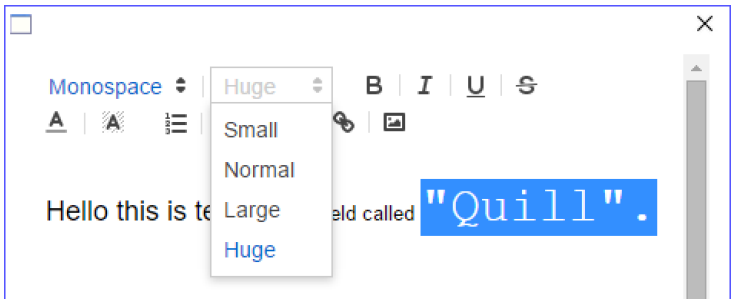
HTML Components for Desktop Applications

Omnis Studio 8.0 gives you the ability to add your own custom HTML components to your window classes, which are used in the thick client, that is, for desktop based applications. (*Note this feature does not relate to the JavaScript components for creating web and mobile apps* – this feature refers to using HTML based controls on *window classes.*)

By adding HTML controls to your window classes you can enhance the UI in your desktop applications and accelerate your development projects – you can obtain many different types of ready-made HTML based components from third-party sources, from simple data controls, to date selectors, to full gantt charts, with the richness and interactivity you would expect to see in web-based applications.

Omnis HTML controls can be thought of as thick client external components implemented using HTML, JavaScript and CSS – in effect, you can use any browser based technology to implement the new HTML controls. In order to add an HTML based control to a window class there is a new browser object, **OBrowser,** which can, in this case, be used to display a single HTML based control in a window class (the OBrowser object can also act as a regular browser for displaying web pages, which is discussed later in this section).

The Omnis HTML controls themselves are located in the **'htmlcontrols'** folder in the main Omnis Studio program folder. Each control has its own sub-folder in the htmlcontrols folder, and the name of this folder is used as the name of the control, e.g. the files for a control named List would be placed in a folder called List. The Omnis tree contains some example HTML controls which you can use for testing, or as a basis for creating your own custom controls. These examples, such as the Quill component which provides a basic text editor, are not supported controls in their own right, so we don't recommend using them as-is in your applications.



To add an HTML control to a window class, you need to add the OBrowser object to the window, which is available in the External Components group in the Component Store, and set its **$urlorcontrolname** property to the name of the control – this property displays a droplist containing the names of all the available controls installed into the htmlcontrols folder in your Omnis development tree, including any you have added. Having added the control to your window you can set its properties in the same as way as any other Omnis component. See the 'Adding HTML controls to your window' topic later in this section.

NOTE: If the OBrowser object is not visible in the Component Store you can enable it via the Studio Browser. To do this, select your library in the Studio Browser, click on the **External Components** option, scroll the list of Component Libraries, select the OBrowser Library option, set the Pre-Load Status to "Starting Omnis" and click on OK.

Creating Omnis HTML Controls

Each Omnis HTML control can be comprised of a number of files which are placed in a folder in the 'htmlcontrols' folder in the Omnis program folder. The main file for each control is an HTML file which is named **<control name>.htm.** For example, if you want to create a control called "quill" you need to create an HTML file called 'quill.htm' which is placed in a folder named 'quill' within the 'htmlcontrols' folder. The .htm file is the file loaded into the browser control (set using $urlorcontrolname) when the control is used on your window.

In addition, there may be a JSON file named **<control name>.json** in the control's folder which defines the htmlcontroloptions row. Plus, the control folder can contain other resources needed as part of the control implementation, such as JavaScript files, CSS files, image files, and so on. The control .htm file typically has links to these other resources.

When you have added the correct files to the relevant folder the control will be ready to use and add to the window classes in your application. To deploy your application, you will need to add the same files and folder structure to the Omnis runtime tree.

Third-party HTML controls

As well as creating your own controls using HTML, JavaScript, and CSS, you can obtain many ready-made controls from third-party sources, either on an open source or paid-for basis. The HTML code for a control needs to be embedded into the Omnis compatible HTML template which is required to load a control into the browser object in Omnis.

<control name>.htm

The .htm file for a control defines a jOmnis object, its various callbacks, and the HTML content for the control itself, embedded at the place marked "…control-specific contents…" in the HTML code. The file has the following structure:

```
<html>
<head>
   <title></title>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
   <script type="text/javascript" src="../omn_list_base.js"></script>
   <script type="text/javascript" src="../omnishtmlcontrol.js"></script>
   <script>
      jOmnis.callbackObject = {
         omnisOnLoad: function () {},
         omnisSetOptions: function(options){},
         omnisCssChanged:function(){},
         omnisSetData: function (value) {},
         omnisGetData: function () {},
         omnisGetCurrentLine: function(){},
         omnisGetSelection: function(){},
         omnisSetFocus: function () {},
         omnisTab: function() {},
         omnisGetDraggedData: function (event) {},
         omnisDropHilite: function (hiliteLine, destinationId) {},
         omnisDropUnhilite: function () {},
         omnisGetDropLine: function (mouseX, mouseY) {},
```

```
        omnisDoScroll: function(scrollDirection, scrollAmount){},
        omnisOnWebSocketOpened: function() {},
        omnisOnWebSocketClosed:function() {}
      };
   </script>
</head>

<body style="margin:0;" class="omnishtml">
   …control-specific contents…
</body>
</html>
```

Note that the **omnishtml** class could be applied to another element rather than the body.

There are two JavaScript files used at the start of the page.

- **omn_list_base.js**
  provides an object that allows you to manipulate an Omnis list represented in JSON. You can read this file in order to see how to use it.
- **omnishtmlcontrol.js**
  provides the interface between JavaScript and the OBrowser component. This file is not minified, so you can read it to understand it.

<control name>.json

The JSON <control name>.json file defines the options and properties of your HTML control. The control's JSON document has the following structure:

```
{
    "data": "multi",
    "options": {
        "selectedcolor": 255,
        "margin": 0,
        "mainiconid": 1710,
        "title": "The title",
    },
    "optionsDescriptions": {
        "selectedcolor": "Description for selectedcolor option",
        "margin": "Description for margin option",
        "mainiconid": "Description for main icon id",
        "title": "Description for title"
    }
}
```

The top-level object has members as follows:

Member	Description
data	This member indicates how data is exchanged between OBrowser and the JavaScript running in the HTML control. It can have three possible values: all: This means that when OBrowser wants to get the current value of the control from JavaScript, it will retrieve all of the data. NOTE that all is the default data handling mechanism if the control does not have a .json file. single: Applicable to controls for which $dataname is a list. When OBrowser wants to get the current value of

Member	Description
	the control from JavaScript, it will retrieve just the current line. This is useful for single select lists that do not modify the data. multi: Applicable to controls for which $dataname is a list. When OBrowser wants to get the current value of the control from JavaScript, it will retrieve the current selection state and the current line. This is useful for multiple select lists that do not modify the data. single and multi provide optimized data handling for lists that do not modify the data.
option	Each member of options is an option that can be used to modify the behaviour of the control. The options object defines the members and their default values. Each member must be a simple type (number, boolean or string). The initial value of $htmlcontroloptions for a new control is the value of this object. If you edit the json file to include new options, OBrowser should detect them and update $htmlcontroloptions. When OBrowser sends the options to JavaScript it sends the current value of $htmlcontroloptions. If the option name ends in “color” then the value stored in the options object is an integer RGB value. When OBrowser sends this to JavaScript it converts it to a CSS color value. If the option name ends in “iconid” then the value stored in the options object is an integer icon id value. When OBrowser sends this to JavaScript it converts it to a file URL. When using an icon from an icon datafile or #ICONS, the file URL uses the same PNG file as that used for the JavaScript client. You will need to manually delete the PNG in the html/icons folder if you edit the icon, to allow Omnis to re-generate the file.
optionsDescriptions	There should be a member in this object for each member of options. The values are used as tooltips when editing $htmlcontroloptions using the property manager.

The Callback Object

omnishtmlcontrol.js creates an instance of an omnishtmlcontrol object called jOmnis. Your page sets the callbackObject member of jOmnis, to allow the omnishtmlcontrol object to communicate with your control implementation.

Your callback object can contain its own members, but do not use ‘omnis’ as the prefix of a member name, since we use omnis to identify methods provided by the callback object that may be called by omnishtmlcontrol.

Methods called using $callmethod are members of the callback object.

The following table describes the omnis-prefixed methods you need to provide in a callback object. Each member is marked as mandatory or optional to indicate if you must provide an implementation.

Method	Description
omnisOnLoad	Mandatory. Called when the browser onLoad event occurs. Use this to perform initialization.
omnisSetOptions	Optional. Called to set the options for the control. It has a single argument, which is a JavaScript object containing the members defined in the control .json file. When first loading the control, omnisSetOptions is called before CSS is applied and before set data. Once the control is loaded, omnisSetOptions can be called again if the application assigns $htmlcontroloptions.
omnisCssChanged	Optional. Called after the CSS in the omnishtml class has been added to the page, or updated. When first loading the control, omnisCssChanged occurs after omnisSetOptions, but before set data. Once the control is loaded, omnisCssChanged can be called whenever a property that contributes to the omnishtml class is changed.
omnisSetData	Mandatory. Called to set the data for the control. One parameter, the data. If the data is a row variable, then the parameter is a JavaScript object with a member for each row column; the data types of the members must be simple types (character, boolean, integer, number). If the data is a list, then the parameter is an instance of omnis_list (see omn_list_base.js for details). Otherwise, the data is a value of a simple type. Note that for character data, OBrowser converts Omnis line endings (\r) to suitable line endings for the HTML control (\n) before calling omnisSetData.
omnisGetData	Mandatory for controls which have a data mode of "all". Called to get the data from the control. Returns the data for the control. If the data is a row variable, returns a JavaScript object. It must have the same definition as the originally set row. If the data is a list, the return value must be an instance of omnis_list. Otherwise the return value must be a simple type. Note that for character data, OBrowser converts browser line endings (\n) to Omnis line endings (\r) in the returned data.
omnisGetCurrentLine	Mandatory for controls which have a data mode of "single" or "multi". Returns the current list line.
omnisGetSelection	Mandatory for controls which have a data mode of "multi". Called to get the list selection from the control. Returns an array of integers, with a member for

Method	Description
	each list line. A member is zero if the line is not selected, one if the line is selected. The array entry for line 1 is at array index zero.
omnisSetFocus	Optional. Called when the control receives the focus. You may need to focus an element when this method is called e.g. this.elem.focus().
omnisTab	Optional. Passed a single parameter, the JavaScript keydown event. Called when a tab occurs. This gives the control the opportunity to tell Omnis to tab out of the control. omnishtmlcontrol provides default behaviour based on HTML tabindexes - you can override the default by providing omnisTab. To tell Omnis to tab out of the control, omnisTab calls jOmnis.tabOutOfControl with a single Boolean argument which is true to perform a shift tab.
omnisGetDraggedData	Optional. If your control supports drag data, you provide this callback to let Omnis obtain the data being dragged. The return value is the dragged data, or null if nothing can be dragged. You can either return text, or a list, or a row. There are helper methods in jOmnis: makeDraggedDataList and makeDraggedDataRow to assist with the latter two return types.
omnisDropHilite	Optional. Called to highlight the control when it is a possible drop destination during drag and drop. Two parameters: hiliteLine, destinationId hiliteLine is Boolean, true if the $hiliteline property is set for the control. destinationId is the drop destination id, typically a line number returned by omnisGetDropLine if highlighting lines is supported. There is a helper method in jOmnis, that can be used to highlight the entire control. For example: jOmnis.appendDefaultHiliteDiv(document.body); appendDefaultHiliteDiv returns the appended div, so you can remove it from the DOM when unhighlighting.
omnisDropUnhilite	Optional. Called to remove drop highlighting from the control. No parameters.
omnisGetDropLine	Optional. Called when $hiliteline is true, to determine the line over which the pointer is positioned. The return value is the line number (destination id). It takes 2 arguments: mouseX, mouseY These are the current pixel coordinates of the pointer.

Method	Description
omnisDoScroll	Optional. Called to scroll the control while the pointer is over its edges during drag and drop of data. It takes 2 arguments: scrollDirection, scrollAmount scrollDirection is an eScrollDirections value (see omnishtmlcontrol.js) that identifies the direction to scroll. scrollAmount is the maximum number of pixels by which to scroll. Scroll by this amount, if scrolling is desired.
omnisOnWebSocketOpened	Optional, and not normally needed. Called when the socket between OBrowser and the HTML control opens.
omnisOnWebSocketClosed	Optional, and not normally needed. Called when the socket between OBrowser and the HTML control closes.

Sending Events

jOmnis contains APIs that allow you to send events to $event:

API	Description
sendClickEvent	jOmnis.sendClickEvent(lineNumber) Generates evClick with pLineNumber set to lineNumber.
sendDoubleClickEvent	jOmnis.sendDoubleClickEvent(lineNumber) Generates evDoubleClick with pLineNumber set to lineNumber.
sendControlEvent	jOmnis.sendControlEvent(infoObject) Generates evControlEvent with pInfo set to the row corresponding to infoObject.

Debugging

You can edit config.json (see the later configuration section) to enable debugging of your control.

On Windows, once you have set a remote debugging port, open Chrome and navigate to http://127.0.0.1:nnnn where nnnn is the remote debugging port.

On macOS, right click on the control and select inspect element. Note that the web inspector window that opens does not work that well with our window ordering.

Drag Object Support

Due to the way pointer events work with the control, when you enable drag object or drag duplicate, Omnis displays a small bar at the top of the control to enable it to be dragged. There is a property, $dragobjectbarcolor that you can use to set the color of this bar.

Reloading HTML Controls

You can use the $reload() method with an Omnis HTML control to reset it to its initial state. $reload() also automatically redraws the control, so its data will also be set. Using $reload like this is useful when debugging your JavaScript.

Tooltips

The tooltip property for an Omnis HTML control is applied once when the control is created, and it is not re-evaluated. If you want to change the tooltip after creation, you must use $callmethod to provide an interface to change a title attribute in the HTML.

Adding HTML controls to your window

Having created or obtained an HTML control and placed it in the htmlcontrols folder, you can use it in a window class in your library. There are also a number of example controls for you to use as well, such a simple List and Quill, a basic text editing control. Once the control is placed on your window it can be used and updated in the same way as any other control.

Locate the OBrowser object in the External Components group in the Components Store and add one to your window class (if the Browser object is not visible in the Component Store you can enable it using the External Components option when your library is selected in the Studio Browser). Open the Property Manager and set the $urlorcontrolname property to the name of the control – the droplist for this property contains the names of all the available controls installed into the htmlcontrols folder.

When you select a control its properties will be displayed in the Property Manager, including the following properties.

$dataname

An Omnis HTML control can be data bound. The dataname can be a list, a row, or any other simple non-binary type. This makes the control behave like any other data bound control, with one small exception that improves performance. Omnis does not redraw the control when it gets the focus. The only real consequence of this is that you need to explicitly call $redraw in order to update the control.

$disabledefaultcontextmenu

The underlying browser has its own context menus e.g. a TEXTAREA with spell checking enabled has clipboard menu items, as well as spelling suggestions etc. The underlying browser menu is considered the default context menu, and you can disable this using the $disabledefaultcontextmenu property.

$htmlcontroloptions

An Omnis HTML control may have a row of options that can be used to configure its behavior. You can consider these to be custom properties. The property manager has a droplist button for this property, which opens the editor for these options. Options with names ending in color or iconid are edited using the color picker or select icon dialog respectively. The fixed column at the left of the editor has tooltips that display descriptions for the members of the options row.

If the control does not have any options, this property is read-only.

$applycss and $cssextra

You can optionally apply a CSS class named omnishtml to the Omnis HTML control (note that the control needs to explicitly use omnishtml - if it does not, then apply CSS will not have any affect). Set $applycss to kTrue if you want to use the omnishtml class.

The omnishtml class contains entries for various other properties of the browser object (OBrowser): $backcolor, $backalpha, $textcolor, $align, $fontsize, $fontstyle, $font. $font uses the JavaScript client font table entry corresponding to the window font.

In addition, OBrowser also concatenates the value of $cssextra to the end of the omnishtml class e.g. you could set $cssextra to “text-decoration:line-through;text-transform: uppercase;”.

$dragmode

You can set $dragmode to kDragData to enable drag from the control. This only works if the particular control has been designed to support drag data. Drag and drop uses the standard Omnis drag and drop messages.

$hiliteline

List controls that accept dropped data can be configured to highlight individual lines during drag and drop. You can set this property to true, to indicate that you want highlight line behavior, but you will only get that behavior if the Omnis HTML control currently being used supports it.

In addition, when evDrop occurs for a control with $hiliteline set to true, the pDropId event parameter identifies the area of the control over which the drop is to occur, either a line number or ident (when $hiliteline is true), or zero if the control is not list-based (or $hiliteline is false).

$callmethod()

Omnis HTML controls can have methods. You can use the $callmethod() method to call a method within a control or one of the standard callbacks.

- $callmethod(cName,vParam)
  Calls method cName in the control object, passing parameter vParam; returns a unique id for this call. The method runs asynchronously and sends the evCallMethodDone event to the control on completion (see Events below)

```
; method gets the data from the example Quill control
Do $cinst.$objs.quill.$callmethod('omnisGetData') Returns iID
; event method for Quill control assigns the data returned to a var
On evCallMethodDone
   If pID=iID
      Calculate iData as pData
   ; Do something
   End If
```

Ports

The OBrowser component operates on the same port as the Omnis Server which is either assigned dynamically or via the Omnis-server property. For debugging HTML controls, OBrowser opens another port for WebSocket communications between htmlcontrols and Omnis. This is $serverport + 1, or 6912 if $serverport not set. This can be overridden by setting "obrowser > htmlControlPort" in config.json.

On Windows only (if "obrowser > canDebug" is true in config.json), OBrowser opens another port to allow remote debugging of the web content. By default this is port 5989, but can be overridden by setting "obrowser > remoteDebuggingPort" in config.json.

Events

Omnis HTML controls have some basic events, such as single and double click, but they can have custom events.

evCallMethodDone

The evCallMethodDone event is triggered when a $callmethod() is completed: it has three parameters in addition to pEventCode:

Parameter	Description
pUniqueId	The unique id that was returned by $callmethod(). This associates this event with the original call.
pReturn	The return value of the control method. NULL if an error occurred - see pErrorText for details.
pErrorText	Text describing the error.

evClick and evDoubleClick

Omnis HTML controls can generate standard click and double click events, with the pLineNumber event parameter.

evControlEvent

Omnis HTML controls can generate custom events. Each custom event sends evControlEvent. This has one parameter in addition to pEventCode:

Parameter	Description
pInfo	A row containing information about the event. If the control generates more than one type of control event, a column in this row can identify the event type

Browser Component

The Browser Component, called **OBrowser,** is an external component control for thick client window classes that provides the ability to add custom HTML controls to window classes (as described above), as well as embed web pages into your thick client windows. Even though the OBrowser control is a fully featured web browser, it is really only intended for targeted use with specific web pages, rather than use as a general web browser (you should note that there is no sandbox support). You could use it in your application, for example, to present the end user with information in a web-style layout such as a Help system or FAQ, or you could embed a landing page that is hosted on your website.

OBrowser is available for both Windows and macOS (Cocoa); there is no Linux implementation. On Windows, it uses the Chromium Embedded Framework as the underlying browser, whereas for Cocoa it uses the built-in Cocoa WebView. Both the underlying browser implementations provide good support for HTML5 and CSS3.

To add a Browser control to your window class, locate the OBrowser object in the External Components group in the Components Store and drag it onto your window (if the Browser object is not visible in the Component Store you can enable it using the External Components option when your library is selected in the Studio Browser.).

Setting web pages

To navigate to a page in the Browser Component, set the property $urlorcontrolname to a full URL with a prefix either http://, https:// or file://, such as ‘http://www.omnis.net’. Note that in design mode, OBrowser will navigate to the page (assuming there is a network connection if required), but the page will not respond to clicks, keyboard input etc.

Once a page is open in a runtime window, the user can interact with the page as would be expected, although an attempt to open a popup window fill fail. The property $urlorcontrolname does not change while the user navigates through pages. Instead, the read-only property $currenturl contains the URL of the page currently open.

Two more read-only properties provide more state information:

- $cangoforward
  If true, you can use $forward() to move to the next URL in the browser back-forward list.
- $cangoback
  If true, you can use $back() to move to the previous URL in the browser back-forward list.

Methods

OBrowser has the following methods related to using it as a web browser:

Method	Description
$forward()	$forward() navigates forwards to the next URL in the browser back-forward list. Returns a Boolean, true for success
$back()	$back() navigates backwards to the previous URL in the browser back-forward list. Returns a Boolean, true for success
$reload()	$reload() reloads the currently displayed URL. Note that if you want to re-open the assigned $urlorcontrolname, you need to re-assign the original value to $urlorcontrolname.
$startdownload()	$startdownload(iDownloadId,cDestPath) starts the file download with id iDownloadId, storing the file at cDestPath. You must execute either $canceldownload() or$startdownload() in response to the evBrowserStartDownload event. Typically you would prompt for a file path, and then call $startdownload().
$canceldownload()	$canceldownload(iDownloadId) cancels the file download with the specified iDownloadId. You can call this in response to evBrowserStartDownload, to cancel the attempted download. You can also call this any time between calling $startdownload() and receiving evBrowserFinishedDownload.
$setdataurl()	$setdataurl(vData,cMediaType[,cWidth,cHeight]) assigns a data URL for the supplied data, with the specified media type, to $urlorcontrolname. vData can be either: binary - the data represented by the URL or another type - OBrowser converts the data to character if necessary, and then UTF-8, to become the data represented by the URL. cMediaType is a MIME type specifying the type of the data e.g. text/plain or image/png. cWidth and cHeight are optional. They are only relevant if vData is binary, and the data is an image. In this case, the data URL represents an img of the specified cMediaType, sized using the CSS sizes width and height e.g. $setdataurl(image, "image/png", "100%", "100%")

Events

OBrowser generates various events, described in the following sections.

evBrowserLoadStateChange

Sent to the control when it starts or ends loading its content. This event has one event parameter in addition to pEventCode:

Parameter	Description
pLoading	If true, the control is loading content.

evBrowserFrameLoadError

Sent to the control when an error occurs while it is loading a frame. This event has three event parameters in addition to pEventCode:

Parameter	Description
pUrl	The URL being loaded
pFrame	The browser frame. Empty means the main frame
pErrorText	Text describing the error

evBrowserOpenUrl

As mentioned earlier, an attempt to open a popup window will fail. This event is sent to the control when a navigation action by the user wants to open a URL in a new browser window. This event has one event parameter in addition to pEventCode:

Parameter	Description
pUrl	The URL for which opening a popup will fail

evBrowserStartDownload

Sent to the control before starting a file download. Your code must respond by calling one of two OBrowser methods described later: $startdownload() or $canceldownload(). This event has four event parameters in addition to pEventCode:

Parameter	Description
pDownloadId	An integer that identifies this download request
pSuggestedName	The suggested name for the file
pMIMEType	The MIME type of the file
pUrl	The URL of the file to be downloaded

evBrowserDownloadProgress

Sent to the control periodically while a download is in progress. This event has three event parameters in addition to pEventCode:

Parameter	Description
pDownloadId	An integer that identifies this download request
pTotalBytesExpected	The total number of content bytes expected. -1 if the total is unknown
pBytesReceived	The number of content bytes received so far

evBrowserFinishedDownload

Sent to the control when a download has finished. This event has two event parameters in addition to pEventCode:

Parameter	Description
pDownloadId	An integer that identifies this download request
pErrorText	Either empty, meaning the download completed successfully, or error text describing why the download failed

Configuration

The OBrowser object has a section in Omnis configuration file (config.json) named "obrowser". This has entries as follows:

Entry	Description
clearCacheWhenLoaded	Windows only. Boolean. True if the Chromium Embedded Framework cache is cleared when oBrowser is first loaded. The cache is in a sub-folder of the Omnis data folder: chromiumembedded\cache
clearLocalStorage WhenClearingCache	Windows only. Boolean. True if HTML5 local storage is cleared while clearing the cache.
logSeverity	Windows only. Integer. Chromium Embedded Framework log level: 0: Default logging 1: Verbose logging 2: Info logging 3: Warning logging 4: Error logging 99: Logging disabled When logging is enabled, the log is written a file in a sub-folder of the Omnis data folder: chromiumembedded\log\cef.log
remoteDebuggingPort	Windows only. Integer. The TCP/IP port number on which the debugger listens, set to 5989 by default. Set this to zero to disable debugging.
candebug	macOS only. Boolean. If true, context menus for the control allow you to open the developer tools.
htmlcontrolsFolder	Character string. By default, the HTML controls are located in the htmlcontrols folder in the Omnis program folder. You can override this by providing a full pathname in this entry.
defaultHtmlcontrols FolderInDataFolder	macOS only. Boolean. Default is false. If you set this to true, Omnis looks for the htmlcontrols folder in the data folder rather than the program folder (provided that the htmlcontrolsFolder member is absent or empty). This allows you to add your own HTML controls and the Omnis Runtime app to remain code signed.
messageTimeout	Integer. The timeout in tics (1/60th second units) for synchronous communication between OBrowser and the HTML control. Not all communication is synchronous, but certain messages (e.g. get data) need to be. This defaults to 60 (a second, which should be more than enough). When you are debugging your control you may want to make this much larger, to give yourself time in the debugger.

High Definition Displays

With the introduction of Retina displays on Mac desktops and laptops, and 4k displays widely becoming the standard for Windows based computers, support for high definition displays has been introduced in Omnis Studio 8.0. This means that when you run the Omnis Studio SDK on a high definition display the Omnis IDE will be displayed in high definition, and likewise version 8.0 Runtimes will be capable of running your desktop applications in high definition. This enhancement should benefit your application development in general, when working in the Omnis IDE, and your end user applications will see the improvements when run on the latest Mac computers or 4k displays on Windows 8.1.

Omnis detects the resolution of the display it is running on and scales accordingly. On macOS, the new Cocoa based Omnis scales automatically. The coordinates and dimensions used in Omnis and your applications are now virtual 72dpi values. These values are scaled automatically to retina display coordinates.

On Windows, Omnis now determines the physical DPI of the display, and then scales by a factor of 2 if the physical DPI is 192 or greater. Therefore, coordinates and dimensions become virtual 96dpi values. In addition, scaling performed as a result of using the existing $designdpimode property always occurs using the virtual 96dpi if the physical DPI of the display is 192 or greater. Note that Omnis does not support per-screen DPI.

Compatibility Issues

If you for some reason you wish to maintain the scaling mechanism available in previous versions, you can set a new configuration item in config.json, in a new “windows” group, called “highDPIaware”. If set this to false, Omnis will not use the new scaling mechanism, and will run at a virtual 96dpi, and will be scaled automatically by Windows (as it does in 6.1.x and earlier).

It is very likely that small cosmetic changes will need to be made to some of the objects in your libraries when they are scaled to 192dpi using the new mechanism. Font heights and text widths do not always scale proportionately and therefore may need to be adjusted accordingly. You should test your applications thoroughly on a high definition display and make any necessary updates or adjustments to fonts, etc.

Tab strip

The tab strip has been modified to display correctly on HD displays.

- The tab strip now has square tabs with a single pixel border. The edge is now only 1 pixel wide.
- $overlap has been removed.
- The tab strip now has $extraspacing (default zero) to space the tabs if required.
- The tab strip now has $bordercolor, used as the edge color and tab border color.
- $ditherbackground has been removed.

HD Icons and Graphics

To support the increase in definition in the Omnis IDE, we have updated many of the icons and graphics with higher resolution images. In order to support high definition images in your own applications, e.g., pushbutton icons or graphics used in forms and windows, you will need to add higher resolution versions of any icons and update any other graphics. In general, your icon images need to be 2x the size of a standard icon image for display on HD displays and Omnis will try to use the HD version if they exist.

Icon Data files and Icon sets

The existing method of storing icons in #ICONS or an Icon data file and assigning the numeric Icon ID ($iconid) to controls will continue to work, but this is only useful for 16x16 pixel icon images. If you run your application on an HD display and your library uses an icon data file or #ICONS, Omnis will try to use a 32x32 icon (if it exists and the icon page is marked as containing 32x32 icons), in place of the corresponding 16x16 icon. If a 32x32 image does not exist in your icon data file or #ICONS, the existing 16x16 image will be used which may have a very poor visual appearance on newer screens and devices. In order to support high definition 16x16 icons you will need to create a new version of each image at 32x32 pixels and import each one into the icon data file or #ICONS into the 32x32 section on the same icon page using the same icon IDs.

If you have used 32x32 or 48x48 pixel icons in your libraries, and you wish to display them on HD displays, then you will need to adopt the use of **Icon Sets** (separate image files in a folder) which support icon images up to 96x96 pixels (i.e. 2x the largest 48x48 icon size). This feature was introduced in Studio 6.0 and uses icon images that are either 1.5x or 2x the size of standard resolution icon images. Icon sets are supported in JavaScript Client *remote form classes* and on *window classes* in the thick client – you cannot use Icon sets in the old 'Web Client' or 'iOS Client' plug-in (note these plug-ins are not supported in Studio 6.1 or higher).

When Omnis references an Icon ID it will *first search for the icon in an icon set,* then it will search #ICONS in the current library and then any other icon datafiles.

Creating and using HD Icon Images

You can create HD icons in any third-party image editing software and place the images directly in the Omnis tree, in the folder named 'iconsets'. The icon or image files must be saved using the PNG file type and placed in a sub-folder of the 'iconsets' folder in the main Omnis product tree (note this is not the same icons folder in the root of the Omnis tree which contains the built-in icon data files). Each sub-folder represents what is called an **Icon Set** which is a named collection of icons. The name of the sub-folder in the icons folder becomes the name of the icon set which will then appear in the icon selection dialog (this name is specified in $iconset, a new library preference, see below). Note that an icon set cannot be named 'datafile', 'lib', 'studio', or 'studioide' since those names are already used in the icons folder.

Image File names

Each image file within an icon set must conform to the following naming convention:

```
<text>_<id>_<size><state>_<resolution>.png
```

- <text> is the name of image. This string is used in the icon picker dialog when you set an object's $iconid in the Property Manager.
- <id> is the positive integer id to be used as the icon id. It can be in the range 1 to 10000000.
- <size> is the CSS pixel size of the image, i.e. the resolution independent size of the image, meaning that for all resolutions of the same image this has the same value.

The value of <size> has the form <width>x<height>, where the values 16x16, 32x32 and 48x48 are special values since they correspond to the standard icon sizes supported by Omnis.

- <resolution> is the factor by which the pixel density is greater than a standard monitor and is one of the following:
  "_2x" for HD devices such as the Retina display
  "_15x" for some devices e.g. certain Android phones that have a 1.5x pixel density.
  an empty string is the default and is for standard resolution devices, equivalent to _1x

Any files (or folder names) that do not conform to the naming conventions are ignored. Example file names are:

pencil_1657_16x16.png	Normal state 16x16 icon with ID 1657 for standard resolution devices
pencil_1657_16x16_2x.png	Normal state 16x16 icon with ID 1657 for HD resolution devices
check_1658_32x32c_2x.png	Checked state 32x32 icon with ID 1658 for HD resolution devices (see below)

Note that the image file names are case insensitive and they must be unique across all platforms and file systems (that is the case of file names is ignored).

Check Boxes Icons

As a special case you can implement icons for different states in check boxes and radio buttons.

- <state> is the checked, highlighted, or normal state of the icon for multi-state icons and can be one of the following:
  an empty string for the normal state of the icon
  “c” is the checked state of the icon
  “h” is the highlighted state of the icon
  “x” is the checked highlighted state of the icon

Studio IDE icons

If you are unsure about the icons you need to create and the file naming, you can examine the icons in the ‘iconsets/studioide’ folder – here you will see the different size image files and their naming required for each icon used in the Studio IDE.

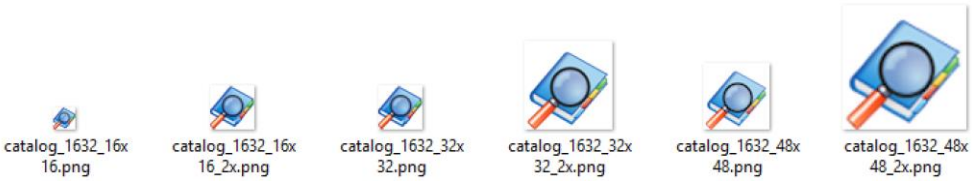


Image Scaling

You do not have to create an icon image for all resolutions, although it would be advisable to do this for the best effect. Omnis will use an icon image closest to the resolution being referenced, scaling as appropriate, and as with all image scaling it is better to force Omnis to scale an image down rather than scale it up. In this case, you may like to provide the highest possible resolution image for your icons and allow Omnis to scale the images to display the lower resolutions, but the scaling may produce unexpected results.

When the JavaScript Client connects, it sends its resolution to the Omnis App Server. This allows the server to use the appropriate icon when setting iconid properties in server methods.

Non-standard Size Images

You can create images with a size other than the standard sizes (16x16, 32x32, 48x48) by creating the image at a non-standard size and including the image size in the file name when the file is saved. For example, you can create an image 100x200 pixels and name it something like “mygraphic_1688_100x200.png”. (Existing users should note that this is the equivalent of an ‘Icon Page’ in the existing icon support.)

Setting the Icon set in your library

Libraries have a preference called $iconset ($libs.LIB.$prefs). This is the name of the icon set to be used when resolving icon ids for the JavaScript client and the thick client in the current library. When using this library, and when looking up an icon for the thick client or JavaScript client, Omnis will search for icons within this icon set before following the current icon search path for the library. In this case icons present in the icon set will take precedence over those in #ICONS, omnispic.df1, etc.

Setting the Icon ID for objects

When you set the $iconid of an object using the Property Manager, the icon set for the current library will be shown in the Icon picker dialog ($iconset must be set for the library for the icon set to appear) allowing you to select one of the icons in the set. You can select the icon required and the Icon ID will be assigned to $iconid for the object.

Errors

Any errors created while setting the icon ID for objects are sent to a file called iconsetlog.txt located in the html folder.

Assigning a URL for images

When you set the $iconid of a JavaScript client object you can also assign a URL. In server methods, if the value being assigned is a character value that contains a "/" character then Omnis treats it as a URL generated by the iconurl function (meaning that it can contain alternative icon files for the different client resolutions, and also that the server will pick the correct icon for the client resolution).

In client methods, if the value being assigned is not an Icon ID (a literal integer or integer + icon size constant) then Omnis treats the value as a URL generated by the iconurl function on the server, and the client picks the correct icon for its resolution.

You could generate the required URLs with the iconurl() function (see below) in the $construct() method of your remote form, and store them in an instance variable list which could then be used in client executed code to assign the correct image to each object.

Image handling for tree lists

For the JavaScript Tree control, the iconid column is now an iconurl column, and the $iconurlprefix property is now redundant although existing libraries that use $iconurlprefix will continue to work. Instead, the iconurl column should be defined to be of type character, and it should be populated using a server-only function, iconurl(iconid), which returns a URL string containing the name of the image file or a semi-colon separated list of file names if an icon exists in more than one resolution. This enables the client to pick the correct icon for its resolution.

Deploying HD Icons

You need to copy your icon sets and images files to the Omnis App Server when you want to deploy your web or mobile app. If the icons are not copied to the Server tree they will appear to be missing from your app.

Standalone Client Apps

Note that for standalone apps the icons needed for your mobile app will be bundled in the SCAF. If any icons change on the Omnis App Server they will be updated on the client when the standalone application files are updated.

Studio IDE and Runtime Icons

Some of the icons used in the Omnis Studio IDE are located in the 'studioide' icon set (other icons used in the IDE are embedded in the Studio Browser itself or are in Omnispic). You may not use the icons in the studioide icon set in your own libraries without the appropriate license: see the license text file in the studioide folder. The studioide icon set is not included in the runtime version of Omnis Studio, and instead replaced with a folder (icon set) called 'studio' containing various icons required for the Print Preview window, Print Destination dialog, and so on. These can be included in your product tree when you deploy your application.

Exporting Icons from an Icon Datafile

You may want to use some existing icons located in an Icon Datafile as separate files and either add to or replace some of them with higher resolution versions. To enable you to export existing icons there is a tool in the Tools>>Add Ons menu, called the 'JS Icon Export' tool, which is available in the 'Web Client Tools' dialog (scroll to the bottom of the list of Web Client tools). The 'JS Icon Export' tool will export all the icons in a

selected Icon Datafile and place them in a folder in the ‘iconsets’ folder, applying the correct image file names. The $iconid property of a control will now reference the external image file and not the icon datafile image.

Auto Updates

You can now perform updates or any other changes to your Omnis application or folder structure upon restarting Omnis by adding a script to the Omnis data folder. You can use the Auto Update feature to update any file in the Omnis Studio tree, including the Omnis executable or program file itself. Under Windows there is one exception – the studiorg.exe file cannot be updated.

To enable the Auto Update feature, write a batch file under Windows called **update.bat,** or on macOS or Linux create a bash script called **update.sh,** and add it to the Omnis data folder, i.e. the folder containing the Studio, Startup, and Welcome folders.

When Omnis starts up it will:

1. Execute the update script automcatically at startup before loading any external components, externals or libraries.
2. If the call to run the script is successful, Omnis then deletes the update.bat/sh file.

When running on Windows, Omnis incorporates a request to run this as part of the existing UAC support implemented via studiorg.exe. In this case, you will get a UAC prompt if the update script needs to run, or if the registry needs updating for some reason, or if both updates and registry updates are required.

The Windows batch file or Unix script must have Execute permissions set in order to run. You can do this in the Properties of the file or via the file system. To do this in your code on macOS or Linux you can use the $setunixpermissions() fileops function:

```
If sys(6)= 'U'   ;; Mac OS or Linux
     Do fileops.$setunixpermissions(
        scriptPathName,'-rwxr--r--' ) ;; set file to execute
End if
```

Example

The following example shows typical commands that could be used in a batch script; the commands download two new xcomps from a server (xcomp1.dll and xcomp2.dll), and store them in a folder specified by con(sys(115),pathsep(),updates,pathsep(),xcomp):

```
copy /y <studio data folder path>\updates\xcomp\xcomp1.dll   <studio program
   folder path>\xcomp\xcomp1.dll
del <studio data folder path>\updates\xcomp\xcomp1.dll


copy /y <studio data folder path>\updates\xcomp\xcomp2.dll   <studio program
   folder path>\xcomp\xcomp2.dll
del <studio data folder path>\updates\xcomp\xcomp2.dll
```

When a path has a space or spaces in it, e.g. Program files, the path should be enclosed in quotes:

```
“C:\Program Files\Omnis Software\OS8.1UPGRADETEST\xcomp\Dummy.dll”
```

Segmented Control

The **Segmented Control** is a new JavaScript control that displays a number of segments or buttons that you can use for navigation or as a toolbar within your web and mobile apps. You can assign an icon and text to each segment and you can detect which segment has been clicked.

The segmented control provides a series of 'segments' arranged horizontally and equally sized to fill the available space, while each segment can contain text and/or an icon. You can optionally show the selected segment in a highlighted state, which is useful if you are using the segmented control as a navigation control.

You can use the segmented control as a toolbar, docking it to the top or bottom of its container by setting its $edgefloat property to one of the kEFposn… values.

Properties

The Segmented Control has the following properties, together with the standard properties for a JavaScript control.

Property	Description
$currentsegment	The number (1 - $segmentcount) of the current segment (this specifies the segment affected by segment specific properties). This can also be changed in a design view by clicking on another segment of the design component. The current segment will be shown with a red outline while the component is selected.
$segmentcount	The number of segments (must be at least one).
$segmentenabled	If true, the segment is enabled and generates a click event when the user presses it.
$segmenticonid	The icon displayed on the current segment. Set to 0 for no icon;
$segmenttext	The text displayed on the current segment.
$displaystyle	A kJSSegmentStyle… value. Controls the appearance of the segments (whether text should be above icon or vice-versa).
$showselectedsegment	If true, the currently selected segment will be shown in a highlighted state. See $selectedcolor & $selectedtextcolor. If false, the highlighted appearance will still be shown while segments are being clicked, to give the user feedback of the click.
$selectedsegment	The number (0 - $segmentcount) of the currently selected segment. If 0 no segment will be selected.
$selectedcolor	The background color of the currently selected segment, or of the segment currently being clicked.
$selectedtextcolor	The text color of the currently selected segment, or of the segment currently being clicked.
$bordercolor	Controls the color of the segment divider lines, as well as the control's border.

Property	Description
$backcolor	Controls the background color of the segments.

Events

An evClick event is generated when one of the segments or buttons is clicked and the pClickedSegment event parameter returns the number of the button clicked.

List Pager

There is a new property in some of the JavaScript **List** and **Grid** components, called **$pagesize,** that allows you to display the lines in the list or grid as separate pages, to improve the user experience when navigating lists or grids with a large number of lines. When assigned an integer value, the $pagesize property forces the list or grid to be sub-divided into a number of scrollable pages, and a set of page number buttons, as well as forward and back buttons, are displayed under the list or grid box which allows you to 'page through' the lines in the list or grid. The default value is zero which means no list pager is displayed.

The $pagesize property has been added to the JavaScript **List, Data Grid, Complex Grid,** and **Native List** (when not using grouped lists). The value assigned to $pagesize specifies the number of list lines displayed in each page, and therefore the total number of lines in the list, divided by the value of $pagesize determines the number of pages in the list or grid field as well as the number of buttons displayed in the pager panel.

Note that setting $pagesize does not reduce the amount of data sent to or fetched from the server – the full list data is sent to the client, and the setting of $pagesize is only used for displaying the list or grid with the pager element.

Changing the Pager's Appearance

The appearance of the pager, such as the color of the buttons, numbers, and arrows, cannot be controlled using standard component properties. However, if you wish to customize the appearance of the pager, you can do so by overriding the associated CSS classes. These classes are named 'omnis.pager<-xxx>' and are listed in the omnis.css file. Do not edit the classes in omnis.css, rather you should override the classes by adding your own version in the user.css file found in the html\css folder in your Omnis development tree.

See the *Creating Web & Mobile Apps* manual for more information about using the JavaScript List and Grid components.

Worker Objects

Push Connections

Support for *Push Connections* has been added to Omnis Studio to allow more control when data is returned to the client when using the Omnis worker objects, such as the SQL Worker objects. In this specific case, you could start off a long query using a SQL

worker on the server, and then push the response to the client when the results are ready, updating any instance variables in the remote form.

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

Creating a push connection

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

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

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

Utilizing REST

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

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

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

then the client scripts will convert this to:

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

for the push connection. You can see the URL used for push connections by using browser debugging tools.

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

Remote Form Method

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

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

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

acknowledgement of pushed data received by the client, so pushed data should not go missing.

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

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

Miscellaneous Enhancements

JSON Objects

Omnis now allows you to manipulate JSON arrays of objects, mapping them to and from Omnis list variables. There are two new static functions in the OJSON external, that work with a single level JSON array, where each array element is an object, and each object has members which are only simple types (integer, number, boolean, string). The functions are:

- OJSON.**$objectarraytolist**(*vData*[,*&cErrorText*])
  Parses vData (binary (UTF8/16/32) or character). vData must be a JSON array of objects containing members with simple types. Returns a list representing JSON. Returns NULL and cErrorText for an error
- OJSON.**$listtoobjectarray**(*lList*[,*iEncoding*=kUniTypeUTF8,*&cErrorText*])
  Writes a list with simple columns to an array of objects; returns JSON with specified encoding (UTF8,UTF16BE/LE,UTF32BE/LE or Character). Returns NULL and cErrorText for an error

When writing a list to an object array, OJSON validates the list, and returns an error if the Omnis data type of a column value is not suitable.

When parsing an array of objects, OJSON validates the data as it parses it, to make sure it is a single array of objects containing only simple types. In addition, OJSON adds columns on the fly, and if a column already exists makes sure the data in the JSON is of the same type as the already added column. This works best when all entries in the array are objects with identical members.

Edge Float Constants

The constants that specify the behavior for floating edges, kEFruntimeLeftRightCenter, kEFruntimeTopBottomCenter and kEFruntimeAllCenter, have been renamed to kEFcenterLeftRight, kEFcenterTopBottom, kEFcenterAll respectively. They now also function during design mode, and can also be assigned to objects on thick client windows, including both foreground and background objects on thick client windows.

Web Services

The $sendhttpcontent() method can now be used to send character data (converted to UTF-8 before sending) and list or row data (converted to JSON before sending). In addition, the method has a new optional second argument, bChunk, which defaults to kFalse (the current behaviour). When true, bChunk formats the data as a chunk (removing the need to call the formatchunk() function). This improves performance a little, and also allows you to handle web servers which automatically chunk the response (e.g. Tomcat). A call to $sendhttpcontent with empty data and bChunk passed as kTrue must be used to terminate the content.

Note that Swagger 2 is now supported for REST based web services: see the *Software Support and Compatibility* section for more information.

HTTP client workers

The HTTP client worker now supports client certificates. There are two new properties of the HTTP client worker object:

- **$keystorepath**
  The path of the Java key store file containing the client certificate to use for HTTP connections. If this property is not empty, $shareconnections is ignored and treated as false
- **$keystorepassword**
  The password for the keystore file identified by $keystorepath

To use these properties, import the client certificate into the key store identified by $keystorepath. Depending on the setup, you may also need to import the server certificate into the omnisTrustStore file.

An example script (setup_client_auth.bat) showing how to set up tomcat to test this locally, and an example tomcat configuration server.xml, is in the secure folder in the main Omnis folder.

Screen Report Fields

Screen Report Fields have two new methods that allow you to zoom or search the screen report. Once the screen report has been sent to the field you can use $zoom() to scale the report from 25% to 200%, and the $searchreport() method allows you to search for the specified text within the screen report content.

Zoom

The method $zoom(*iZoom*) zooms the screen report field, where iZoom can be positive (indicating a percentage between 25 and 200% inclusive), or 0 meaning zoom to fit, or negative (-1 to -7) where -iZoom indexes the 7 standard zoom factors from smallest to largest.

Search

The method $searchreport(*cText*[,*bIgnoreCase*=kTrue,*bNext*=kTrue]) searches the report for cText. Further calls with the same cText and bIgnoreCase search for the next (bNext kTrue) or previous (bNext kFalse) match. Empty cText clears the search.

There is an event which works in conjunction with $searchreport (needed because the search occurs in a background thread). The event enables you to manage next and previous buttons, and status text. The next and previous buttons are assumed to start in disabled state. The event evReportSearchStatus is sent to the report field when the report search status changes: this has one event parameter pReportSearchStatus which is a row with 4 columns, as follows:

next	If true, search next can be enabled as there is another search result later in the report
prev	If true, search previous can be enabled as there is another search result earlier in the report
count	The count of search results
index	The 1-based index of the current search result

Icon Sets

Omnis now looks in up to three folders for icon sets, which are processed in the order stated below. If the same icon set is included in another folder, after it has already been found, it is ignored in subsequent folders and an error written to iconsetlog.txt. You should therefore avoid having the same or similar icon sets in multiple folders to avoid any confusion.

1. A folder named ‘iconsets’ in the program folder.
2. A folder named ‘iconsets’ in the data folder, if it is different to the program folder.
3. The ‘html/icons’ folder, as in previous versions.

The file iconsetlog.txt is created in the Studio folder rather than the html folder.

When using a web server for deploying your application, the icon sets must still be in the html/icons folder in the web server tree, even if they are in one of the other folders in the studio tree.

IMAP

The *IMAPListMessages* external command has a new parameter that allows you to request named RFC822 message headers to be included in the list of messages – the headers are returned as a space-separated list of header names, e.g. "Subject From". The command stores the headers in consecutive columns of the List, starting at column 10, in the order of the header names. These new additional columns need to be defined as character.

Multi-line Fields

Thick client multi-line entry fields have a new property called $linehtextra which is the number of extra pixels added to the height of each line of text displayed in the field. Its value is restricted to 0-255, but in practice you would probably use a small value, just to space out the text a little more than the default.

Rich Text Editor Control

The **Rich Text Editor** JavaScript Control can be used in a remote form to allow the end user to edit and format text using HTML formatting styles including bold and italic, as well as ordered and bulleted lists. For this version the control is now based on **Quill,** which means it should perform well with modern browsers and mobile/touch devices.

The properties and behavior of the Rich Text Editor are more or less the same as the control provided in previous versions with a few minor updates or additions. The $dataname property is an instance variable that stores the editor’s content formatted as HTML, while $plaintextname is a variable that receives a plain-text version of the editor’s contents.

If you have used the Rich Text Editor control on any remote forms in your application, you need to open and save those forms in design mode to convert the control to the new version.

Assigning Text Properties

Note that assigning values to the component’s text properties will set the editor’s ***default*** values for that property, as best it can.

- **$font**
  maps directly to the editor’s font droplist, and will set the default value accordingly.
- **$textcolor**
  will attempt to set the default text color to one of the colors in the toolbar’s color palette. If there is not an exact match, it will add the color as another tile in the palette.

- **$fontsize**
  will set the default font size to the closest match. Set to **13** for the 'Normal' font size as default.

External Class Editor

$editor and $editordata are now properties of all class types except system tables – in previous versions these properties were only available for object classes. The properties allow you to specify your own editor and to access the data for an Omnis class. The definitions of these properties are:

- **$editor**
  The name of the add-in tool library used to edit the class
- **$editordata**
  Editor data stored with the class. Typically used by the library identified by $editor

By default, these properties are not visible in the Property Manager, therefore to make them visible, edit the show_properties item under "properties" in the config.json file:

```
"properties": {
    "show_editor": true
},
```

In addition, you should note that $editordata will only appear in the Property Manager when used in conjunction with the Notation Inspector.

$editor overrides the default editor for a class. Omnis calls $exectool for the specified add-in library, passing it a single parameter which is an item reference to the class.

Note that the specified editor is not used when using find and replace – instead, the normal editor for the class opens.

There is a Tech Note TNID0007 on the Omnis website that shows how you can create an alternative schema editor.

Subforms

End users can now open or close (expand or collapse) subform set panels using a single click – in previous versions, a double-click was required to open or close subforms. This is enabled by making the title bar on the subform behave like the minimize button, and therefore the title accepts single clicks. The **kSFSflagMinButtonIsTitle** flag has been added for the 'subformset_add' action. The new flag only applies when kSFSflagAutoLayout is specified.

When in auto layout mode, not using single open and not using open minimized, you can now indicate that a form is to be opened minimized modes by prefixing its class name with the ~ (tilde) character. This means that when you open a number of subforms in a subform set, you can specify which subforms will open minimized (collapsed).

Datafile Browser

The Omnis Datafiles browser is not now displayed in the Studio Browser by default: to display it you need to enable it under **Options** in the Developer Hub in the Studio Browser.

Omnis Window Title

sys(233) now returns the title of the main Omnis application window.