What's New in Omnis Studio 10.1

Omnis Software

September 2019 (updated Sept 2020 Rev 27575)

47-092019-03

The software this document describes is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. Names of persons, corporations, or products used in the tutorials and examples of this manual are fictitious. No part of this publication may be reproduced, transmitted, stored in a retrieval system or translated into any language in any form by any means without the written permission of Omnis Software.

© Omnis Software, and its licensors 2020. All rights reserved.
Omnis® and Omnis Studio® are registered trademarks of Omnis Software.

Portions © Copyright Microsoft Corporation.
Regular expressions Copyright (c) University of Toronto.

© The Apache Software Foundation. All rights reserved.
This product includes software developed by the Apache Software Foundation (http://www.apache.org/).
Specifically, this product uses Json-smart published under Apache License 2.0
(http://www.apache.org/licenses/LICENSE-2.0)

The iOS application wrapper uses UICKeyChainStore created by http://kishikawakatsumi.com and governed by the MIT license.

Microsoft, MS, MS-DOS, Visual Basic, Windows, Windows Vista, Windows Mobile, Win32, Win32s are registered trademarks, and Windows NT, Visual C++ are trademarks of Microsoft Corporation in the US and other countries.

Apple, the Apple logo, Mac OS, Macintosh, iPhone, and iPod touch are registered trademarks and iPad is a trademark of Apple, Inc.

IBM, DB2, and INFORMIX are registered trademarks of International Business Machines Corporation.

ICU is Copyright © International Business Machines Corporation and others.

UNIX is a registered trademark in the US and other countries exclusively licensed by X/Open Company Ltd.

Portions Copyright (c) The PostgreSQL Global Development Group
Portions Copyright (c) The Regents of the University of California

Oracle, Java, MySQL, and J2SE are registered trademarks of Oracle Corporation and/or its affiliates

SYBASE, Net-Library, Open Client, DB-Library and CT-Library are registered trademarks of Sybase Inc.

Acrobat is a registered trademark of Adobe Systems, Inc.

CodeWarrior is a trademark of Metrowerks, Inc.

This software is based in part on ChartDirector, copyright Advanced Software Engineering (www.advsofteng.com).

This software is based in part on the work of the Independent JPEG Group.
This software is based in part of the work of the FreeType Team.

Other products mentioned are trademarks or registered trademarks of their corporations.

Table of Contents

About This Manual

This document describes the new features and enhancements in Omnis Studio 10.1 Rev 27575, 10.1 Rev 26652, 10.1 and 10.0.

Please see the Readme.txt file for details of bug fixes and any release notes in Omnis Studio 10.1 Rev 27575.

Software Support, Compatibility and Conversion Issues

Serial Numbers and Licensing

You will require a new serial number to run Omnis Studio 10.1. Contact your local sales office to buy a license or obtain an upgrade serial number under your current support program.

Library and Datafile Conversion

Converting 10.0.0 Libraries

******** IMPORTANT NOTE: ********
ONCE A LIBRARY HAS BEEN OPENED WITH OMNIS STUDIO 10.1 IT CANNOT BE OPENED WITH STUDIO 10.0.x.

Converting 8.x or earlier Libraries

Omnis Studio 10.1 will convert existing version 8.1.x, 8.0.x, 6.1.x, 6.0.x and 5.x libraries – THE CONVERSION PROCESS IS IRREVERSIBLE.

IMPORTANT: *IN ALL CASES, YOU SHOULD MAKE A SECURE BACKUP OF ALL OMNIS LIBRARIES AND OMNIS DATAFILES BEFORE OPENING THEM IN OMNIS STUDIO 10.1.*

Disclaimer: *Omnis Software Ltd. disclaims any responsibility for, or liability related to, Software obtained through any channel. IN NO EVENT WILL OMNIS SOFTWARE BE LIABLE FOR ANY INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES HOWEVER THEY MAY ARISE AND EVEN IF WE HAVE BEEN PREVIOUSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.*

Method Editor: Code Conversion

There has been a major rewrite of the code editing part of the Method Editor (in Studio 10.x), which means you can now enter Omnis commands and code using freetype. Due to these major enhancements, there has been a number of enhancements or changes in the Omnis *programming language* and *command syntax*. Therefore, when you convert an Omnis library to Studio 10.x (from Studio 8.x or earlier), your Omnis code will be converted to the new syntax.

IMPORTANT: you should note that once you convert your library and *start using the new free-type Code Editor, you cannot revert back to the old editor in Studio 10.1:* that is, the old interface for modifying methods, using point-and-click, *has been removed from this version.*

See ‘Library Conversion’ under the Code Editor section in this guide for more information about the changes made to the Omnis code syntax during library conversion.

Java Legacy Integration

Oracle has changed the way it licenses Java. Therefore, in order for you to avoid the ongoing use of Java in connection with Omnis Studio, we no longer provide support for various Java files in the Omnis Studio 10.x tree and consequently we have removed various Omnis libraries or features that rely on Java. Any Java-dependent features will no longer appear in Omnis Studio 10 and will only be shown or supported when the relevant files are reinstated back into the Omnis tree. By doing this now, we have allowed you to utilise Java supported features in Omnis Studio by choice only. We urge you to check the Oracle website for details about how Java is licensed and the changes to licensing they have made.

Some of those Java-dependent libraries or features in Omnis Studio that have been removed have been superceded by newer technologies and we encourage you to switch to those for future development. For example, support for the old OWEB Worker Object external commands has been removed and we have replaced all the commands with a new set of commands in the OW3 Worker Object command set (e.g. the POP3 and FTP commands), as well as adding new support for Cryptography, Hashing, and JavaScript (node.js) worker objects.

The following Java-related files and features have been removed or support for them has changed:

- **Java folder**
  The java folder and its contents has been removed from the Omnis Studio development, server and runtime trees; you will need to install Java for any Java-dependent features to work in Omnis Studio
- **JDBC DAM**
  The JDBC DAM (damjdbc) has been removed from Omnis Studio (xcomps) and will no longer appear in the SQL Browser.
- **Java Objects**
  The javaobjs and javacore libraries have been removed; the Reset Java Class Cache hyperlink in the Studio Browser is therefore not shown, and will only appear if the JavaObjs Library is put back in the Omnis tree and loaded
- **Web Services**
  The old SOAP based Web Services library (wsc.lbs in the startup folder) has been removed, and it will no longer appear in the Studio Browser: you should use the new REST based Web Services that do not depend on Java; it is also possible to use SOAP using node.js via the new JavaScript Worker Object
- **Web Worker Objects – Web & Email communications**
  The old OWEB Worker Objects external (oweb in the xcomp) and their associated commands (FTP, SMTP, etc) have been removed and will no longer appear in the IDE (e.g. Code Assistant): you should use the new OW3 based Worker Objects that do not depend on Java. In addition, the OWEB external contained a number of static methods that have been moved to OW3: see below.
- **Java Options**
  The $usejavaoptions and $javaoptions properties no longer appear in the Property Manager and Code Assistant, and will only appear if the JavaObjs Library is put back in the Omnis tree and loaded.

If you wish to continue to use any of the Java-dependent files in Omnis Studio you need to install Java and place any Java-dependent files we used to provide back into the Omnis tree. Please contact Technical Support to obtain the Java files, or look on our developer website (https://developer.omnis.net/) under General Information.

OWEB Static Methods

A number of static methods (functions) in the OWEB external have been moved to the OW3 external command package. You are urged to change your code to use the new methods. You should change your code to use OW3.$method() rather than OWEB.$method().

The OWEB methods affected are:

$makeuri()	$makeuuid()	$unescapeuritext()
$escapeuritext()	$gethardwareid()	

Context Menus & $active

Context menus in JavaScript Remote forms previously only opened if $enabled for the control was kTrue. In Studio 10.x, they are now opened if $active of the control is true: $active is a new property added to all JavaScript components. This may have changed the behavior of your context menus on certain controls, so you are advised to examine any event handling code in your application that opens context menus. See the section in this manual about the new $active property for more information.

IE 9 & 10 Support

Omnis Studio 10.1 does not support IE 9 & 10 for the JavaScript Client. Microsoft ended support for these versions in January 2016. You should also note that support for IE 11 will be dropped from future versions of Studio 10.2 onwards.

Sybase DAM

The Sybase DAM in Studio 10.x has been modified to work with the FreeTDS – libct client library in place of Sybase Open Client. By exploiting the common heritage between technologies, the libct library allows native connection to Microsoft SQL Server databases as well as Sybase ASE and ASA databases.

We have provided a technote (TNSQ0036) which explains how to use the libct client library. This page also provides downloads of pre-compiled libct libraries for Windows, macOS and Linux.

Omnis 7 Events

The $v3events library preference was removed from Omnis Studio version 10.0, but has been reinstated in this version 10.1 for backwards compatibility; note however, the preference is now only visible in the Property Manager via the library preferences in the Notation Inspector. The $v3events library preference is supported in the VCS for converted Omnis 7 libraries.

First Run Receipts on macOS

By default, if a new version of Omnis is installed it will use any existing user data which already exists that matches the Omnis package name.

To preserve pre-existing user data and allow a new installation of Omnis with a new set of user data, a deployment can use the receipt mechanism.

This is enabled by setting resource 25598 to "1" in the Localizable.strings file for the language used, e.g.

```
"CORE_RES_25598" = "1";
```

If receipts are enabled then when Omnis is first run it will add a unique timestamp to the end of the user data folder name, e.g.

```
~/Library/Application Support/Omnis/Omnis Studio 10 x64_20181005085835
```

and place an associated receipt into a folder with the same name as the Omnis package.

```
~/Library/Application Support/Omnis/Receipt/Omnis Studio 10.0 x64
```

This then ties the timestamped user data with that installation of Omnis.

To provide a clean install of Omnis, the receipt folder needs to be removed, e.g. this could be done via a script as part of any deployment installation process.

This will then generate a new set of time-stamped user data while preserving the old set.

Note: The resource in Localizable.strings should not be edited in an already signed package as this will break the code-signature. A package should be re-signed after the change is made.

What's New in Omnis Studio 10.1 Rev 27575

The following new features have been added to Omnis Studio 10.1 Rev 27575. Please see the Readme.txt file for details of bug fixes in 10.1 Rev 27575.

Deployment

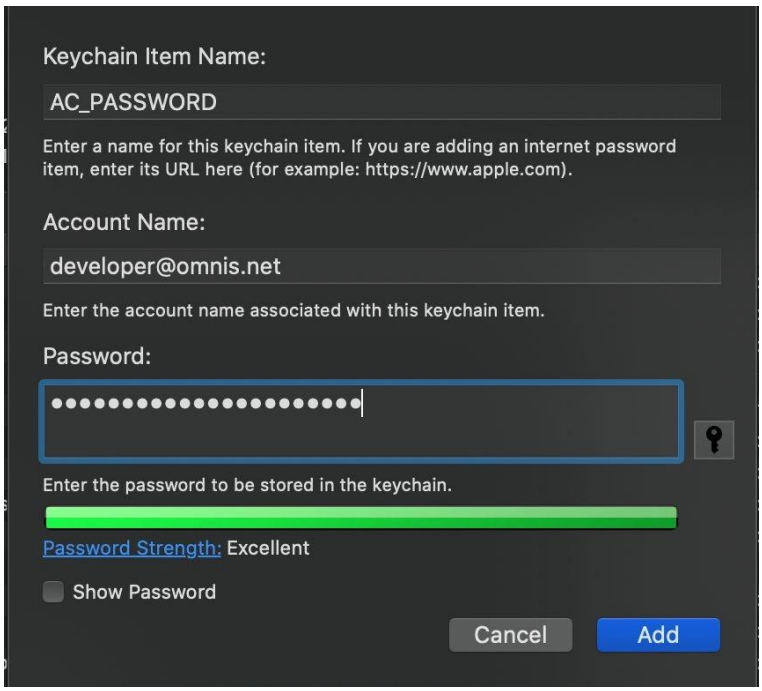
Deployment Tool

Code signing and Notarizing (macOS)

On macOS, you can now **Code sign** and **Notarize** your application using the Deployment tool, which will make deployment easier and faster.

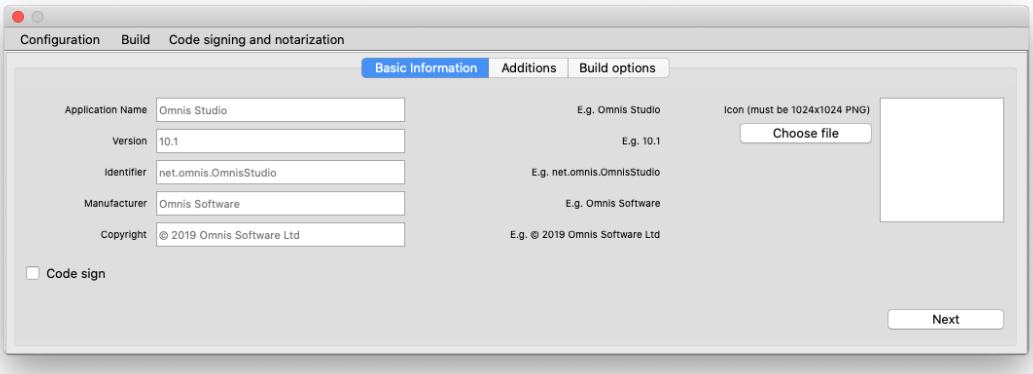
To Code sign and Notarize you need an app-specific password, and this can be obtained by logging into your Apple ID account, via https://appleid.apple.com/account/manage and selecting "Generate password…" under APP-SPECIFIC PASSWORDS. Once generated, copy your app-specific password.

Next, create a new Keychain Access password with the name AC_PASSWORD and paste the password you just copied as the password and your Apple developer email address for the account, as follows:



You will also need a developer certificate in Keychain Access that can be used to code sign and notarize. You can obtain one from your Apple developer account and you can add it to the Keychain Access by double-clicking on the certificate.

Now using the Deplyment tool, you need to check the **Code sign** checkbox on the Basic Information tab to display the Notarize options.



This will open some new options which allow you to enter the Code signing identity. Code signing is required before notarizing and the code signing identity should be the name of your certificate as it comes from Keychain Access, such as:

Developer ID Application: Omnis Software Ltd (XYZ123XYZ123)

Once you enable Code signing, you can also enable the new “Notarize” checkbox which displays a new entry field for your email and a new tab that handles entitlements.

Your email is used when submitting your application to Apple for notarization and you will receive an update from Apple on the status of notarization on the same email.

The new tab, “Notarization options”, can be used to toggle application entitlements, which tell Apple more about what your application will need access to. You need to supply some defaults which reflect what you distribute, and if you need more access, you can just cheque the appropriate option.

Once you are ready to build, just can use the **Build** menu or the shortcut Cmd+B on macOS or Ctrl+B on Windows.

If you Code sign and Notarize as part of the build, Omnis may be unresponsive for quite a time, since a number of things need to happen: existing code signatures are removed, new code signatures are applied, a DMG is built and submitted to Apple, and so on. During this process, you need to leave Omnis open, and once the process is finished you will be prompted if it was successful or not.

Once you have successfully notarized your application, you need to stapler either the DMG or the application bundle before distributing. You can use the “Staple an existing Bundle or DMG” from the Code signing and notarization menu.

You can also check if a DMG or application bundle is successfully stapled via the terminal by executing “stapler validate [path to dmg or app]”.

Omnis ini file (Windows)

You can now control Windows-specific behavior using an 'omnis.ini' configuration file that can be placed at the root level of the writable directory (AppData). You can specify a general section and the following options:

```
[General]
HideStudiorgMessage=0
NoAdmin=0
UpdateFileAssociation=0
```

Hiding the Studiorg Message

HideStudiorgMessage will achieve the same behavior as the test.txt file in the studio folder did (as in previous versions). If HideStudiorgMessage=1 then no message dialog will be displayed about running studiorg when Omnis starts up. If a value is omitted, the default is 0.

Running Omnis as the Current user

NoAdmin can be used to run Omnis for the current user. If NoAdmin=1, Omnis will run with the current user's access level; consequently, it will not attempt to register file associations or event log, and this allows you to run updates (via update.bat) if required. If NoAdmin=0 Omnis will run as the Admin user (the default behavior, as in previous versions).

Setting file associations

UpdateFileAssociation is an existing option and if set to 0, Omnis will not attempt to set file associations.

What's New in Omnis Studio 10.1 Rev 26652

The following new features have been added to Omnis Studio 10.1 Rev 26652:

- **Deployment Tool & RCEdit**
  a new tool that allows you to build and customize the Omnis Runtime tree to deploy your application on Windows and macOS; plus for Windows only, there is a new external component, called RCEdit, that allows you to edit various Omnis program resources, to fully customize or 'brand' your product installation
- **JS Tree Lists**
  JS Tree lists have a new event, evExpandNode, which is fired after the user has expanded a node, every time that node is expanded
- **JS Data Grids**
  The property $tabthroughcells has been added to JS data grids to change the action of the tab key while the focus is on the grid
- **oBrowser**
  Omnis now assigns the port for the OBrowser control dynamically
- **Remote Forms: $loadfinished**
  The $loadfinished method has been added to remote forms to allow you to check when all subforms of a form have been loaded

Deployment

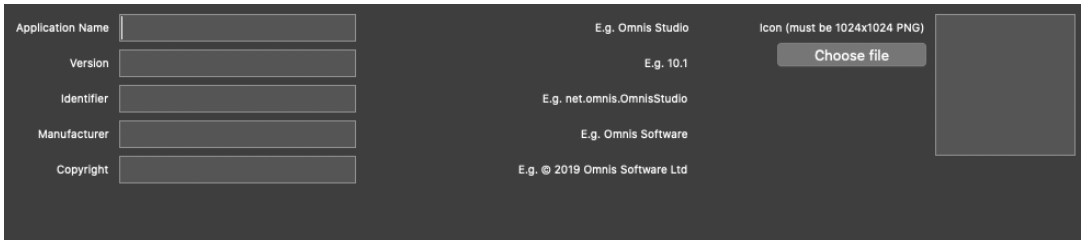
Deployment Tool

There is a new **Deployment Tool** that allows you to build and customize the Omnis Studio macOS and Windows product tree (Runtime installer) suitable for deployment to your customers. On macOS the tool lets you create a bundle, while on Windows it allows you to create a full or split trees. In addition, for Windows only, there is a new external component, called RCEdit, that allows you to edit various Omnis program resources, to fully customize or 'brand' your product installation.

You can open the Deployment Tool using the Tools >> Add-ons >> Deployment tool option on the main Omnis menu bar. The tool is system dependent, so there are minor differences between the capabilities and interface across Windows and macOS which are described below.

macOS

For macOS, you can change the Application Name, Version, Identifier, Manufacturer, Copyright notice and Icon in the first screen:



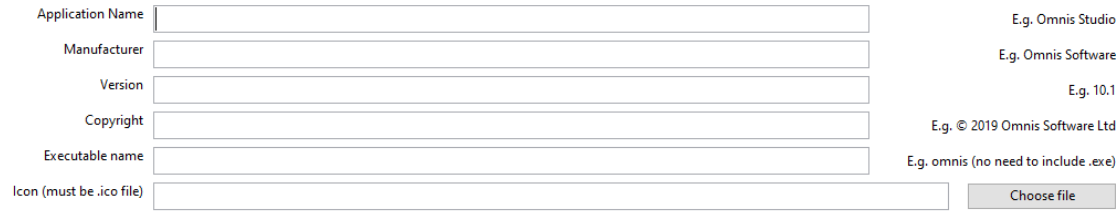
The second screen allows you to specify the bundle's startup folder, iconsets (for the library), xcomp and icons folders, as well as the option to pre-serialise the bundle or add a custom read/write directory.

On the Build options screen you need to select the Omnis Bundle (runtime), a build folder (the path to the folder where you want your output to go), and the version of your Bundle.

You can also select the option to clear the build folder if there is any error in the build process.

Windows

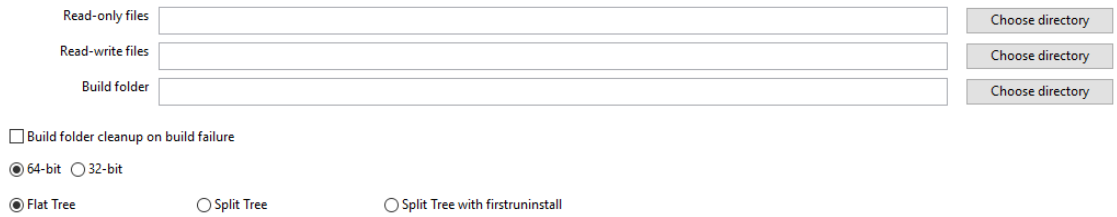
On Windows, you can select a new Application Name, a Manufacturer, Version, Copyright notice, Executable name and the path to an .ico file to replace the Omnis icon.



The second screen lets you specify the bundle's startup, iconsets (for the library), xcomp and icons folders, as well as the option to pre-serialise the bundle or add a custom read/write directory.

On the Build options screen you need to specify the location of the Omnis read-only files (these are usually your Omnis Studio xx.x RT folder in your ProgramFiles), the read-write files (usually the files in AppData), a build folder (path to the folder for the output), and the option for 32 or 64 bit.

You can also select the option to clear the build folder if there is any error in the build process.



On Windows, you can also select the option to build a folder or separate folders providing a Flat tree, Split tree or a Split tree with firstruninstall.

- The **Flat tree** option will output a folder containing both read and read/write files.
- The **Split tree** option will output a folder containing read-only files, to go in the Program Files folder, and a read-write folder containing files for the AppData folder.
- The **Split Tree with firstruninstall** option will output a folder containing read-only files and inside that an additional “firstruninstall” folder, containing the read-write files that Omnis will copy the first time the application will be run.

The **firstruninstall** option allows you install and setup your application without the need to build an installer, which may be quicker or more convenient for your deploy process or product cycle. For example, you could use 7Zip's SFX archive feature to create an executable which simply unarchives itself in such a way to make installers and the complexities they come with unnecessary.

Windows Resource Editor (RCEdit)

There is a new external component called RCEdit that allows you to edit various Windows resources, including the product version, version string, and the icon for the Omnis executable. The external component implements a number of new methods which you can call from your Omnis code.

Setting the application manifest

$setapplicationmanifest(cFile, cManifest) sets the manifest in cManifest to file path cFile. For example:

```
Do rcedit.$setapplicationmanifest("C:\omnis.exe", "C:\folder\newManifest.xml")
   Returns #F
```

Setting a resource string

$setresourcestring(cFile, cResource, cValue) sets resource in cResource to value in cValue for file path cFile. For example:

```
Do rcedit.$setresourcestring("C:\omnis.exe", "1", "This is the new value")
   Returns #F
```

Setting the product version

$setproductversion(cFile, cProductVersion) sets the Windows executable resource “Product Version” to version in cProductVersion for file path cFile. For example:

```
Do rcedit.$setproductversion("C:\omnis.exe", "10.1")
```

Setting the Omnis icon

$seticon(cFile, cIcon) sets the Windows executable icon in file path cFile to .ico file path in cIcon. For example:

```
Do rcedit.$seticon("C:\omnis.exe", "C:\newIcon.ico") Returns #F
```

Setting the program version string

$setversionstring(cFile, cVersion, cValue) sets the version string in cVersion to value in cValue for file path in cFile. For example:

```
Do rcedit.$setversionstring("C:\omnis.exe", "Comments", "Comment version
   string") Returns #F
```

Setting the program file version

$setfileversion(cFile, cFileVersion) - sets the Windows executable resource “File Version” to value in cFileVersion. For example:

```
Do rcedit.$setfileversion("C:\omnis.exe", "10.1.0.0") Returns #F
```

JavaScript Components

JS Data Grid

$tabthroughcells

The property $tabthroughcells has been added to JS data grids to change the action of the tab key while the focus is on the grid. If set to kTrue (default is kFalse), tabbing from a cell which is not being edited selects the next cell, or Shift+tab selects the previous cell.

In addition, setting $hcell or $vcell now triggers edit mode if $autoedit=kTrue.

JS Tree Lists

JS Tree lists have a new event, evExpandNode, which is fired after the user has expanded a node, every time that node is expanded (unlike evLoadNode which is only triggered if the node has no children). This applies to both dynamic and non-dynamic tree lists.

You cannot use $nodedata to load data into the tree list with this new event, it is just a notification and includes the parameters pNodeIdent and pNodeTag. If evLoadNode and evExpandNode are both active, evLoadNode will be fired first, as evExpandNode is fired after the node is expanded.

JavaScript Remote Forms

$loadfinished

The $loadfinished method has been added to remote forms to allow you to check when all subforms of a form have been loaded. The client-executed method is called after all the subforms that belong to the parent remote form instance have finished loading and their $init methods have been called.

Window Components

oBrowser

Omnis now assigns the port for the OBrowser control dynamically. (This change was made to fix issues with OBrowser HTML control port and multiple instances of Omnis.)

The change means you should no longer configure htmlControlPort in the OBrowser section of config.json.

Object Oriented Programming

Object Instances

Object instances created via sub-type now belong to the current task at the point of their creation; this provides consistency with object instances created via $new.

Web and Email Communications

HASH Worker

The maximum key length for a HMAC hash used in the HASH worker has been increased from 32 to 64.

What's New in Omnis Studio 10.1

The following features were added to Omnis Studio 10.1 (released in Sept 2019), including a number of enhancements in the Code Editor, plus some updates for JavaScript Remote forms and various JS controls.

- **Variable Panel in Code Editor**
  The *Variable panel* is a powerful addition to the Method Editor that allows you to view and modify variables *while you debug and step through your code*; as execution pauses, the Variable Panel displays the values of all the current variables, and you can drill down into the hierarchy of objects and variables
- **Code Editor & Code Assistant**
  There are many enhancements in the Code Assistant including: *Method name matching* to allow you to find a method name as you enter code; *Command Keywords* are added to a command automatically when pressing Tab, enabled using a new option in the Line menu; a new option *Copy Value* in the *Variable menu* allows you to copy the value of a variable
- **SQL Worker Lists**
  you will be able to specify that a SQL list or row uses a *SQL Worker Object* of the same DAM type as the SQL session object to perform SQL operations asynchronously in a separate self-contained thread (or synchronously if preferred).
- **Managing Timeouts for Remote tasks**
  Remote Tasks used with the JavaScript Client now have a concept of being 'suspended' to allow greater control over how client connections are managed using the new properties $suspendedtimeout and $suspendconditions
- **Toast Messages**
  There is a new client command to allow you to popup "Toast messages" (small temporary notifications) on the client, similar to Android toast messages
- **New and Updated JavaScript Components**
  The JS Video control has been rewritten to remove its reliance on jQuery, and as a consequence the control has some new properties and events; in addition, the Data Grid, Toolbar, Date Picker, and Tree List JS controls have all been enhanced
- **Line: command**
  There is a new command, *Line:,* which is like the *Text:* command, except that it just adds a single line of text to the text block; there is a new external editor (similar to the JavaScript: and Sta: editors) to allow you to add consecutive sequences of Line: commands
- **OBrowser for macOS**
  The macOS version of OBrowser now uses the *Chromium Embedded Framework* (CEF), which the Windows version of OBrowser already uses; the macOS version of OBrowser now supports the standard OBrowser CEF configuration settings using the cefSwitches configuration item in the config.json (as on Windows)
- **New Window Class Controls & Animation**
  There is a new library and object property, $animateui, that allows you to animate certain window class controls. Tree Lists have the new property, so when enabled the contents of a node will animate when it opens (also used in some parts of the Studio IDE); plus the Tab Strip has some new type constants to animate the tabs. There are two new window class External Components: an iOS-style *Switch* control and a *Multibutton* (both need to be loaded into the Components Store)
- **Trace Log**
  The Trace Log has been added to the Studio browser, available via a new node in

the Studio Browser tree list, which shows the current number of lines in the log; the new view of the trace log behaves the same as the existing trace log (except there is no max lines setting)

Code Editor

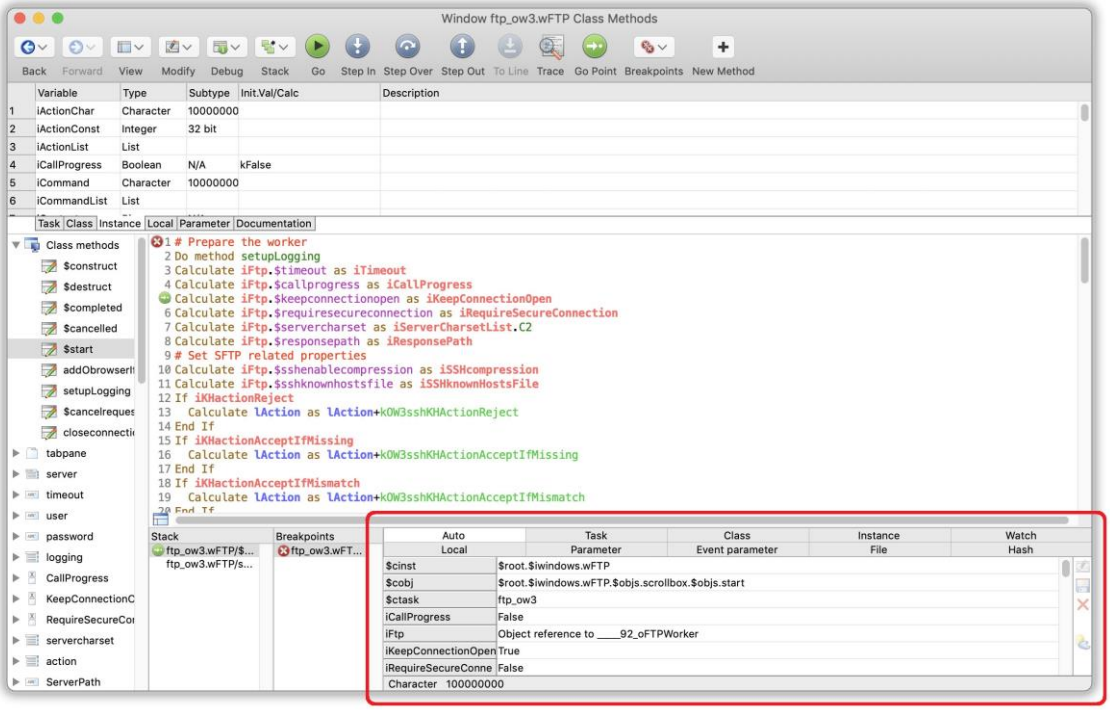
The Code Editor (Method Editor) introduced in Studio 10.0 has had a number of enhancements, including several in response to customer feedback.

Variable Panel

The Variable panel was introduced for Remote Debugging in Studio 10.0 but is now available in the standard Code Editor/Method Editor in Studio 10.1.

The **Variable panel** allows you to view and modify variables while debugging: note it is only populated *when execution pauses,* such as with a breakpoint. After you resume execution, it remains populated (but disabled) for a short time, until either execution pauses again (when it updates) or execution does not pause soon enough (in this case it clears).

When execution pauses, the focus moves to the variable panel. For example, while stepping through code the Variable panel will show $cinst, the task and instance variable values, and the values of any watched variables: see the Variable panel highlighted in red below.



Viewing Variable Data

The variable panel displays a hierarchy of controls that allow you to drill down into the data. Each time the debugger pauses execution, it refreshes each level of the hierarchy until it reaches a level which is no longer valid, e.g. you might drill down into a local list variable, and execution pauses in a different method, so the local list is no longer valid, so in this case the panel will display the local variables of the new method.

In many cases, the panel displays variables in a grid using either the row or list representation of the grid as appropriate. The grid display for a variable or list cell shows a text representation of the value. This may be either its value, or it may be some other representation, e.g. the number of lines in a list, or an object instance name. The grid is read-only, allowing you to use the arrow keys or tab/shift-tab to move around the grid.

As you move around the grid, the current cell is highlighted, and the data type of the current cell is displayed in the status bar below the grid.

Sometimes a cell represents data such as a list or an object – in this case, you can drill down to view the contents of the cell by either clicking on the cell, or by pressing the Return key. After drilling down, a back button appears in the area above the grid, that you can use to navigate to the previous level, or alternatively you can press Backspace.

You can Ctrl/Cmd+click on a cell that would normally drill down, in order to give that cell the focus.

Buttons to the right of the grid enable, disable or check depending on what you can do with the current cell.

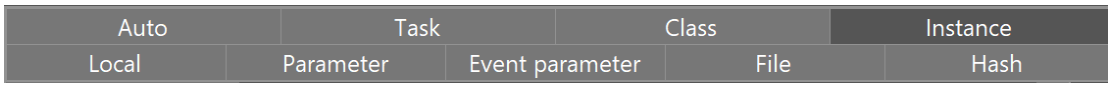
When enabled, you can click on the Modify button, or press the Return key, to edit the variable value. While in edit mode, the remainder of the window disables, apart from Cancel and Save buttons. You can use the Escape key to cancel, and the Return key to save the value (i.e. the key specified as saveModifiedVariable in keys.json): note that the Return key does not allow you to save the variable if it makes sense to add returns to the data being edited.

There is also a button to toggle the current value between NULL and empty.

Top Level Variable Panel

When you first pause execution, the debug window displays the top-level variable panel. This allows you to view Auto, Task, Class, Instance, Local, Parameter, Event Parameter, File and Hash variables. Auto comprises variables identified from the line before the current line (if any), the current line, and up to 2 lines after the current line.

The top of the top-level variable panel allows you to select the currently displayed scope:



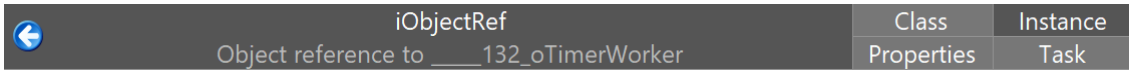
You can either click on a button (heading), or type its first letter when the variable panel has the focus, to display the scope. Save Window Setup will save the current scope.

With the exception of the File scope, each scope displays its variables in a grid. The file scope initially displays a list of file classes. You can then drill down into a file class, in order to view its values.

For task, class and instance variables, the panel shows the values for all levels of the inheritance hierarchy, with the names of inherited variables shown in the inherited color.

Object Variable Panel

When you drill down into an object or object reference, the panel displays properties and/or variables. The top of the panel looks like the following:



In the case of a non-visual object, all the buttons at the right are hidden, and the panel just shows properties. In the case of a sub-classed non-visual object, all buttons are present and enabled. In the case of an object that is not sub-classed from a non-visual object, the properties button is disabled.

As for the top-level panel, you can either click on a button, or type its first letter when the variable panel has the focus, to display the scope.

List or Row Variable Panel

For a row, this is a straightforward grid. When you drill down into a list, the panel initially displays the first 64 lines (or $linecount if less than 64) of the list. **Next** and **Previous** buttons at the top-right of the panel allow you to read more lines:

If you hold the Shift key while pressing the button, the panel reads all data in the direction specified, in chunks until there is no more. While doing this, it may display a working message (if it takes long enough), which you can use to stop any further data being read.

Each time you step, and the variable remains in scope, the panel initially updates with the chunk of data from the start of the current scroll position.

You can modify the current line and selection of the list using the buttons on the variable panel. These prompt for the new current line, or changes you want to make to the selection.

Item Reference Panel

When you drill down into an item reference that has properties (rather than an item which is a reference to a variable), the panel displays the property values of the item. You can use this panel to modify values for which $canassign is kTrue, provided that they are of a suitable data type for editing.

Large Character

Character variables containing more than 128 characters are displayed as their length followed by a preview of the start of the data. You can drill down into the variable, displaying a character variable panel. When you first drill down, this displays up to the first 64k characters. **Next** and **Previous** buttons at the top-right of the panel allow to to read more chunks:

If you hold the Shift key while pressing the button, the panel reads all data in the direction specified, in chunks until there is no more. While doing this, it may display a working message (if it takes long enough), which you can use to stop any further data being read.

Each time you step, and the variable remains in scope, the panel initially updates with the chunk of data from the start of the current scroll position.

If you edit the data, the edit applies to the entire variable value, i.e. the new value comprises any data on the server before the loaded data, followed by the edited loaded data, followed by any data on the server after the loaded data.

Binary

To view and edit a binary variable, you always need to drill down. You are then presented with a hex binary editor grid. When you modify the variable, a button on the right provides various binary editing operations. The binary panel works in a similar way to the character panel, with next and previous buttons.

Picture

You can drill down into a picture variable and edit it.

Boolean

Boolean variable values can be Empty, False or True. These can be set using the variable grid drop list.

Keyboard Shortcuts

Various keyboard shortcuts have been added for the Code Editor or Remote debugger, and a small number of shortcut keys have changed (from 10.0). All the keyboard shortcuts can be viewed or edited in the $keys Omnis preference in the Property Manager or the ‘keys.json’ configuration file.

Modify Class and Modify Methods

The Modify Class and Modify Methods keyboard shortcuts are now configurable in the keys.json file. The new shortcuts are named modifyClass, modifyMethods and modifyFieldMethods and appear in the new "ide" group in keys.json.

modifyMethods and modifyFieldMethods also apply to the window, remote form, and report class editors.

modifyMethods also applies to the class browser.

The default for the modifyThisClass shortcut in the methodEditorAndRemoteDebugger section of $keys has changed to F3.

Clear Method Stack

There is a new keyboard shortcut for Clear Method Stack, which is Alt+K on Windows, or Cmnd+Opt+K on macOS. The new shortcut is clearMethodStack in the remote debugger and method editor group in $keys.

Go point

You can Shift click in the left margin to set the Go point, and there are configurable keyboard shortcuts for Go, Step, Set Go point, etc.

Win & macOS Keyboard Shortcuts

The following keyboard shortcuts are available in Studio 10.1:

Windows shortcut	macOS shortcut	Description	Keys.json item
Alt+A	Cmnd+Opt+A	Replace all in method	replaceAllInMethod
Alt+B	Cmnd+Opt+B	Disable breakpoint	disableBreakpoint
Alt+C	Cmnd+Opt+C	Match case	matchCase
Alt+E	Cmnd+Opt+E	Enable breakpoint	enableBreakpoint
Alt+F	Cmnd+Opt+F	Disable all breakpoints	disableAllBreakpoints
Alt+G	Cmnd+Opt+G	Enable all breakpoints	enableAllBreakpoints
Alt+H	Cmnd+Opt+H	Open Edit helper	openEditHelperDialog
Alt+I	Cmnd+Opt+I	Debugger interrupt	debuggerInterrupt
Alt+J	Cmnd+Opt+J	Set list selection	setListSelection
Alt+K	Cmnd+Opt+K	Clear method stack	clearMethodStack
Alt+L	Cmnd+Opt+L	Set list current line	setListCurrentLine
Alt+M	Cmnd+Opt+M	Toggle read-only mode	toggleReadOnlyMode
Alt+N	Cmnd+Opt+N	Toggle null and empty	toggleNullAndEmpty
Alt+R	Cmnd+Opt+R	Replace next in method	replaceNextInMethod
Alt+S	Cmnd+Opt+S	Save modified variable	saveModifiedVariable
Alt+T	Cmnd+Opt+T	Set breakpoint	setBreakpointCondition

Windows shortcut	macOS shortcut	Description	Keys.json item
		condition	
Alt+U	Cmnd+Opt+U	Duplicate line	duplicateLine
Alt+V	Cmnd+Opt+V	Go to Variables panel	gotoDebuggerVariables
Alt+W	Cmnd+Opt+W	Whole words	wholeWords
Alt+X	Cmnd+Opt+X	Regular expression	regularExpression
Alt+Y	Cmnd+Opt+Y	Side by side	sideBySide
Alt+Z	Cmnd+Opt+Z	Binary edit operations	binaryEditOperations
Ctrl+/	Cmnd+/	Toggle comment	toggleComment
Ctrl+[	Cmnd+[	Move up stack	moveUpStack
Ctrl+]	Cmnd+]	Move down stack	moveDownStack
Ctrl+0	Cmnd+Opt+0	Go to Task variables	gotoTaskVariables
Ctrl+1	Cmnd+Opt+1	Go to Class variables	gotoClassVariables
Ctrl+2	Cmnd+Opt+2	Go to Instance vars	gotoInstanceVariables
Ctrl+3	Cmnd+Opt+3	Go to Local variables	gotoLocalVariables
Ctrl+4	Cmnd+Opt+4	Go to Parameters	gotoParameters
Ctrl+5	Cmnd+Opt+5	Go to Docs panel	gotoDocumentation
Ctrl+6	Cmnd+Opt+6	Go to RESTful panel	gotoRESTfulPanel
Ctrl+7	Cmnd+Opt+7	Go to code panel	gotoCode
Ctrl+8	Cmnd+Opt+8	Go to method tree	gotoMethodTree
Ctrl+D	Cmnd+D	Select word	selectWord
Ctrl+E	Cmnd+E	Execute method	executeMethod
Ctrl+F	Cmnd+F	Find in method	findInMethod
Ctrl+G	Cmnd+G	Find next in method	findNextInMethod
Ctrl+H	Cmnd+H	Replace in method	replaceInMethod
Ctrl+I	Cmnd+I	Insert before	insertBefore
Ctrl+L	Cmnd+L	Go to line number	gotoLineNumber
Ctrl+M	Cmnd+M	Insert method at end	insertMethodAtEnd
Ctrl+N	Cmnd+N	Insert after	insertAfter
Ctrl+R	Cmnd+R	Next error	nextError
Ctrl+U	Cmnd+U	Lower case selection	lowerCaseSelection
Ctrl+Shift+B	Cmnd+Shift+B	Toggle breakpoint	toggleBreakpoint
Ctrl+Shift+C	Cmnd+Shift+C	Clear code breakpoints	clearCodeBreakpoints
Ctrl+Shift+D	Cmnd+Shift+D	Delete selected methods	deleteSelectedMethods
Ctrl+Shift+E	Cmnd+Shift+E	Trace	trace

Windows shortcut	macOS shortcut	Description	Keys.json item
Ctrl+Shift+G	Cmnd+Shift+G	Find previous in method	findPreviousInMethod
Ctrl+Shift+I	Cmnd+Shift+I	Inherit and override method	inheritAndOverrideMethod
Ctrl+Shift+J	Cmnd+Shift+J	Clear variable breakpoints	clearVariableBreakpoints
Ctrl+Shift+K	Cmnd+Shift+K	Delete current line	deleteCurrentLine
Ctrl+Shift+L	Cmnd+Shift+L	Select line	selectLine
Ctrl+Shift+M	Cmnd+Shift+M	Superclass methods	superclassMethods
Ctrl+Shift+N	Cmnd+Shift+N	Show method tree	showMethodTree
Ctrl+Shift+O	Cmnd+Shift+O	Toggle one-time breakpoint	toggleOneTimeBreakpoint
Ctrl+Shift+R	Cmnd+Shift+R	Previous error	previousError
Ctrl+Shift+S	Cmnd+Shift+S	Step	step
Ctrl+Shift+T	Cmnd+Shift+T	Step out	stepOut
Ctrl+Shift+U	Cmnd+Shift+U	Upper case selection	upperCaseSelection
Ctrl+Shift+V	Cmnd+Shift+V	Step over	stepOver
F1	F1	Opens the Omnis Help using the syntax item *under* the pointer	(Not configurable)
F3	F3	Modify this class	modifyThisClass
F5	F5	Go point	go
F7	F7	Fix error	fixError
F8	F8	Modify specified class	modifySpecifiedClass
F10	F10	Method history backwards	methodHistoryBackwards
Shift+F1	Shift+F1	Opens the Omnis Help using the syntax item *under* the pointer	(Not configurable)
Shift+F2	Shift+F2	Set Go point	setGoPoint
Shift+F4	Shift+F4	Pin bottom panel	pinBottomPanel
Shift+F5	Shift+F5	Hide bottom panel	hideBottomPanel
Shift+F6	Shift+F6	Show editor panel	showEditorPanel
Shift+F7	Shift+F7	Show debug panel	showDebugPanel
Shift+F9	Shift+F9	Show variable panel	showVariablePanel
Shift+F10	Shift+F10	Method history forwards	methodHistoryForwards

Method Name Matching

A new keypress has been added to the Code Editor to allow you to search for a method name where the name of a method is required in a line of code.

You can press **Shift-space** after entering a string in the code assistant and any possible matching method names are added to the help list. For example, you could enter: *Do code method* ***test*** and then press Shift-space, and the Code Assistant displays all strings containing "test" that can be used as a method name parameter of Do code method.

For notation, if you enter $test and then **Shift-space,** the code assistant only shows matching strings that are notation (start with $) and contain "test".

Command Keywords

There has been a number of improvements to the handling of optional keywords for commands.

There is a new option on the Code Editor **Line menu,** *Tab Adds Missing Optional Keyword,* which is enabled by default. In this case, pressing Tab for commands that have optional keywords, such as *Do, For* and *Enter data,* the Code Assistant appends the optional keyword(s) to the command, ready for you to enter its parameter(s). If you do not want the keyword added by tab, undo will remove it.

This occurs when the cursor is somewhere in the command, the command does not already have the missing keyword(s), and no characters are selected. For example, pressing tab after entering Do $cinst.$test() will add the "Returns" keyword.

In the case of the *For* and *For each line in list* commands, tab will cause the keywords "from", "to" and "step" to be added in turn.

The state of the Tab Adds Missing Optional Keyword option is saved with the window setup.

Fonts

The Code Editor now supports variable-width fonts (Studio 10.0 only allowed fixed width fonts in the code editing area). Therefore, the various elements of the Code Editor, including the code area and method list, can now use any of the default fonts provided by the current operating system: e.g. on Windows Segoe UI and Consolas are used as the default fonts. You can change the fonts used under the View>>Fonts option: the Reset option lets you return to the default fonts for your OS.

Variable Menu

There is a new option in the Variable context menu, "Copy Value", to allow you to copy the current value of the variable.

Code Conversion

Print report command

The code converter converts a *Print report command* with no instance name, *but with* constructor parameters, by adding * as the instance name (in previous versions this was causing a conversion error). For example, Print report (list) becomes Print report * (list) after conversion, which executes in exactly the same way.

Text: command

The code converter no longer maps open parentheses in Text: command to ['('']. As part of this change, the Code Assistant now behaves differently when you are typing the text for a Text: command. When you type (at the end of the text, the code assistant opens, and displays the options for the Text: command. You can either select one of the options or carry on typing something else. In the latter case, Omnis now treats the characters you type as text rather than options.

Find and Replace

The Find and Replace function in the Code Editor has been improved.

When you execute the Find or Replace commands while the find or replace panel is open, the editor now sets the focus to the find field and selects all the text.

The content of the find field is either:

- The text currently selected in the editor, provided that it is all on one line
- Or if no text is currently selected or the selected text spans lines, and highlight syntax words is turned on, the current syntax word
- Or if no text is currently selected or the selected text spans lines, the current search data.

Inherited Methods

Showing Inherited Methods First

There is a new option on the View menu of the Code Editor, **Show Inherited Methods First,** which allows you to display inherited methods at the top of the methods list in the Code Editor; the option defaults to off which means inherited methods will be shown after all other methods at the bottom of the list, as in previous versions.

In addition, the remote debug server configuration has a new option (Show inherited methods first in method lists) which controls the information returned by the server to the client, and therefore the display in the remote debug window. The remote debug server dialog has been updated to allow this option to be edited.

Editing inherited methods

The F8 shortcut now works for inherited methods. So if you press F8 on the code line Do $inherited.$test() it will load the $test method in the inherited class.

List Field References

The Code Assistant now includes list column names from the current definition of a variable, and assistance for the target of a field reference variable.

Lists can be defined when debugging code, for example, when the method editor is attached to an instance, or when the variable is a class variable. The target of a field reference variable can be determined when execution is stopped at a breakpoint.

Entering Quotes, Braces, and Square Brackets

There is a new mechanism in the Code Assistant to detect situations where automatically supplying the ” (closing quote) after typing an “ (opening quote) makes sense.

Similarly, a } (closing brace) is inserted automatically where it makes sense after typing { (open brace), or a] (closing square bracket) is inserted after you enter a valid calculation after entering an [(opening square bracket); this includes the case when entering a calculation at the end of the parameter for the Sta: command.

Overtyping closing quotes and brackets

Note that when entering a string, if the caret is positioned just before a closing quote, and you type the same quote character, the editor overtypes the closing quote rather than inserting another. The same overtyping will occur with closing braces and closing square brackets.

Construct Parameters

Where possible, the Code Assistant help window now expands “params...” for $add, $open, etc to show the constructor parameters of the class referenced. Omnis identifies the class name that precedes the method name in your code (e.g. classname.$open), and will show the construct parameters for the class.

Copying Code

When you copy text from the Code Editor, Omnis now copies the syntax coloring and other formatting, to allow you to paste the code into a word processor or an email and retain the colors and formatting. The code is copied in HTML format.

Unicode Characters

The handling of Unicode characters >= 0x250 in the Code Editor has been improved.

The Code Editor now selects a smaller font size, if necessary, for all Unicode characters >= 0x250 contained in a string. On retina displays (on Win and macOS), the display of these characters is improved, using the default Code Editor font. On non-retina displays, it may be necessary to increase the font size to get a reasonable display.

Character Constants

The constants kHash (# character), kLeftSB and kRightSB (left and right square bracket) have been added to allow you to insert those characters into text. If you wish to create a constant for double hash, you could initialise a variable with the value con(kHash,kHash).

Inline Comments

When Omnis encounters a space character followed by ## at the end of a string it treats it as the start of the inline comment, so if space## appears in a string it will be treated as an inline comment. To overcome this, you can enter <space>## in a string and it will not be interpreted as an inline comment.

Read-only Mode

The "Read-only mode" option on the Modify menu in the Code Editor has been improved. When this menu is enabled, you can toggle the editor between read-only and write mode using the new keyboard shortcut Alt+M / Cmnd+Opt+M (stored in $keys). The method editor now stores the state of "Read-only mode" with the Window Setup.

VCS and Read-only mode

If you are using the Omnis VCS, the VCS read-only state will override the "Read-only mode" of the Code Editor. In addition, you cannot toggle the state using the Modify menu, and if you perform Save Window Setup, the saved state of "Read-only mode" will be unchanged.

Toggle Comment

Empty method lines are no longer commented out when using the Toggle comment command or Shortcut key: this applies when multiple selected lines may include empty code lines.

Rename Variable

A Rename Variable option has been added to method editor Variable context menu and parameter helper to allow you to rename a variable in your code directly (rather than having to go to the Variable pane); the option applies to class, instance, local and parameter variables.

Variable Descriptions

The variable description is now included in variable value tooltips.

File Class Field & Library Names

When unique field names is true, the Code Editor does not enter a file class name prefix when you enter a file class field/variable name into the Calculate command, for example, for file classes in the same library as the class being edited. There is a new configuration item called 'checkFileClassPrefixBasedOnUniqueFieldNames' to control this behavior; the new item is true by default.

When unique field names is false, checkFileClassPrefixBasedOnUniqueFieldNames requires that you enter a file class name prefix, for file classes in the same library as the class being edited.

In addition, the Code Assistant now only shows file class field names at the top level when unique field names is on; so if unique field names is off, the list just includes the file class names.

And finally, the Code Assistant now includes library names at the top level, to allow references like lib.file.field to be entered, or lib.<library notation> to be entered.

Obsolete Commands

All the obsolete commands were hidden (removed) from the Code Editor in Studio 10.0, however for backwards compatibility some of these commands, listed below, have been reinstated in 10.1. See the appendix for a list of obsolete commands that have been removed from the Code Editor and will be commented out on conversion.

There is a new option on the Filter Commands menu in the method editor, **Command List Can Show Obsolete Commands** (defaults to unchecked), which allows you to view the obsolete commands that have not been removed and are still available in the Code Editor.

To confirm, the following Obsolete commands are no longer removed or commented out during conversion in Studio 10.1, and will continue to work as expected.

Clear task control method OBSOLETE COMMAND

Clear window control method OBSOLETE COMMAND

Disable fields OBSOLETE COMMAND

Enable fields OBSOLETE COMMAND

Hide fields OBSOLETE COMMAND

Queue click OBSOLETE COMMAND

Queue double-click OBSOLETE COMMAND

Queue scroll OBSOLETE COMMAND

Queue set current field OBSOLETE COMMAND

Redraw named fields OBSOLETE COMMAND

Redraw numbered fields OBSOLETE COMMAND

Redraw windows OBSOLETE COMMAND

Send to a window field OBSOLETE COMMAND

Set return value OBSOLETE COMMAND

Set task control method OBSOLETE COMMAND

Set window control method OBSOLETE COMMAND

Show fields OBSOLETE COMMAND

SNA do not perform default action OBSOLETE COMMAND

SNA perform a Cancel OBSOLETE COMMAND

SNA perform a shift-tab OBSOLETE COMMAND

SNA perform a tab OBSOLETE COMMAND

SNA perform an OK OBSOLETE COMMAND

SNA perform command OBSOLETE COMMAND

SNA perform default action OBSOLETE COMMAND

SNA remain on current field OBSOLETE COMMAND

SNA set current field OBSOLETE COMMAND

Hide design & commands menus OBSOLETE COMMAND

Test if command available OBSOLETE COMMAND

Store window OBSOLETE COMMAND

Set palette when drawing OBSOLETE COMMAND

Set return value OBSOLETE COMMAND

During conversion, consecutive *Set return value OBSOLETE COMMAND* ***value*** and *Quit method* commands (the latter with an empty parameter) are combined into a single command *Quit method* ***value*****.** Note that when checking for consecutive commands, Omnis skips comments and empty lines.

Call Method OBSOLETE COMMAND

The *Call method OBSOLETE COMMAND* is converted to the *Do code method* command using the same parameter as the old command.

SQL Worker Lists

Currently, you can define a list or row variable from a SQL class (query, schema or table class), and associate a SQL session object with the variable in order to perform various SQL operations on the list, e.g. populate the list from the database, insert a row into the database.

This new feature allows you to specify that the SQL list or row will use a SQL Worker Object of the same DAM type as the SQL session object to perform SQL operations asynchronously (or synchronously, if preferred). Because the worker can run asynchronously, there are some differences in the way that you can use a table class from which the list or row is defined, compared to the way you use the table class with a SQL session object, as in previous versions of Studio. Specifically, there is less scope to override SQL methods using the table class because of the need to execute the worker in a separate self-contained thread.

Using a Worker in a SQL List or Row

$useworker and $synchronous

If you want to use a worker object with your SQL list or row, you need to assign a new property, $useworker to kTrue. $useworker must beassigned after assigning $sessionobject, and once you have assigned $useworker, you can no longer assign $sessionobject, or access $statementobject (the latter is destroyed if present when $useworker is assigned). $useworker cannot be assigned to kFalse.

In addition, there is a new property $synchronous: if true, and $useworker is true, the worker object for the schema or table instance executes synchronously in the current thread rather than asynchronously in a separate thread. $synchronous defaults to false (meaning use another thread).

In addition, Omnis does not expose the worker properties $waitforcomplete and $cancelifrunning.

$waitforcomplete will always be kTrue, to make sure the application is notified of the success or failure of an operation, and $cancelifrunning is not relevant - the table will not invoke a new request until the previous request has completed - requests are queued by the table instance while the worker is busy processing a request.

Selecting & Fetching Data

Non-worker SQL lists and rows can operate in a nice synchronous manner. So $select() can be used to generate a result set, and $fetch() can be called multiple times to retrieve the result set.

SQL Worker based lists and rows cannot run in this simple synchronous manner, because the result set is generated by the worker in a separate thread. Therefore, worker SQL lists and rows have a new method, $selectfetch that performs both the select and the fetch of the data. It has the following definition:

- **$selectfetch()**
  $selectfetch([bDistinct=kFalse, iMaxRows=1, bAppend=kTrue, cText,…])

Note that $selectfetch() cannot be used with a row variable defined from a SQL class, so if you want to fetch data using a worker you must define a list from the SQL class.

Note also that you cannot override $selectfetch() in a table class. The parameters are as follows:

- **bDistinct**
  Pass this as kTrue to make the worker use a SELECT DISTINCT query rather than SELECT.
- **iMaxRows**
  The maximum number of rows to fetch. Must be between 1 and 10000000 inclusive.
- **bAppend**
  Pass this as kTrue to append the fetched data to the list, kFalse to replace the list contents with the fetched data.
- **cText,…**
  Any further parameters are treated as SQL text and appended to the generated SELECT or SELECT DISTINCT query.

Any errors that are detected before invoking the worker object, result in a call to $sqlerror in the table instance.

After fetching the data, the worker generates a notification to $completed in the table instance.

Inserts, Updates and Deletes

When using a worker, you cannot override $insert, $update or $delete in a table class.

When you execute these methods via a worker, the table instance copies the current values of the affected row (rows for $update) into the parameter list for the worker, and then starts the worker.

Any errors that are detected before invoking the worker object, result in a call to $sqlerror in the table instance.

On completion, the worker generates a notification to $completed in the table instance.

Smart List Methods

When using a worker, you cannot override $dowork, $doinserts, $doupdates, $dodeletes, $doinsert, $doupdate or $dodelete. Also, you cannot call $doinsert, $doupdate or $dodelete.

When you call $dowork, $doinserts, $doupdates or $dodeletes, the table instance generates a single query for each of the relevant operations insert, update and delete. The instance then copies bind variable values into a list, for each set of rows to be inserted, updated or deleted. Finally, the table instance starts the worker with the copied data as its parameters. When the worker completes, the worker generates a notification to $completed, that identifies any rows for which an error occurred, with information about the error.

Note that as soon as you call $dowork, $doinserts, $doupdates or $dodeletes, the smart list updates just before starting the worker

Any errors that are detected before invoking the worker object, result in a call to $sqlerror in the table instance.

Completion Row

The table instance properties $rowsaffected and $rowsfetched are not relevant when using a worker.

$completed in the table instance is passed a row variable parameter with columns as follows:

- **errorcode**
  An error code. Zero means the worker was successfully passed the query and bind variables. Note that the query or queries may still have failed - see the errors column.

- **errortext**
  Error text describing the errorcode.
- **work**
  The list of queries and bind variables that were passed to the worker. This has the usual structure for SQL workers - two columns, query and bindvars.
- **errors**
  If errorcode is zero, this is a list of queries that generated a SQL error of some sort. This has the same structure as the Errors column generated by a SQL worker in the worker completion row.
- **rowsFetched**
  If a call to $selectfetch successfully fetched some rows, this is the number of rows fetched.

JavaScript Remote Forms

Managing Timeouts in Remote Tasks

There is a new mechanism to handle timeouts in remote tasks.

Remote Tasks used with the JavaScript Client now have a concept of being 'suspended' to allow greater control over how client connections are managed. A task may (optionally) be suspended if the web page is sent into the browser's persistent cache, or if the page becomes hidden (e.g. the user switches tabs).

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

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

Suspend Properties

To support this, Remote Tasks have two new properties:

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

The conditions under which the client may suspend are:

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

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

Important Note: The HTML templates have all been updated as part of this enhancement, therefore you need to update any .htm files on your web servers to match, otherwise you will get errors or leak Remote Tasks.

Remote Tasks Events

Remote Tasks have two new events:

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

Remote forms Events

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

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

These methods receive the following parameters:

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

Remote Form Template file

The template .htm files have been updated, so it's important that you update any existing .htm files on web servers/included in wrapper apps etc accordingly.

Toast Messages

Toast messages are small notification type messages that that can be "popped up" in a remote to alert the end user about something: the concept is derived from "toast messages" on Android.

Toast messages are activated using a new client command "showtoast" which displays a message to the user in a small popup which disappears after a timeout, either 5000ms or specified amount.

- Do $cinst.**$clientcommand**("showtoast",row-variable)
  Where row-variable is row(text, [timeout, posX, posY, containerName, fixed, originX, originY, speakMessage, assertive])

The toast message row-variable parameters are:

- **text:** The message text. The container's size will scale with the amount of text. You can use '\n' to insert a new line.
- **timeout:** (Optional) The length of time (ms) the message will be shown for (5000 ms by default).
- **posX:** (Optional) The horizontal position of the toast message in pixels. Centered if not specified. If containerName is specified, this position is relative to the control.
- **posY:** (Optional) The vertical position of the toast message in pixels. Positioned near the bottom of the form if not specified. If containerName is specified, this position is relative to the control.
- **containerName:** (Optional) The name of the control to position the toast message relative to. Options are limited to paged pane and subform controls.

- **fixed:** (Optional) This determines whether or not the toast message will stay in position when the container scrolls (true by default).
- **originX:** (Optional) The toast message's origin that posX references. Possible values are: kLeftJst (default), kRightJst and kCenterJst.
- **originY:** (Optional) The toast message's origin that posY references. Possible values are: kJstVertTop (default), kJstVertMiddle and kJstVertBottom.
- **speakMessage:** (Optional) If true, screen readers will announce the message. This is an accessibility feature to convey information to visually impaired users.
- **assertive:** (Optional) If speakMessage is true, this instructs screen readers whether or not to interrupt current speech.

The originX and originY parameters are used to set the point on the toast message that posX and posY reference. For example, if originX is kRightJst and originY is kJstVertBottom, the bottom right corner of the toast message will be at the position specified by posX and posY.

Push Connections

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

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

Subform Sets

Scroll Position

In previous versions, there were some inconsistencies with where a subform in a subform set was initially positioned relative to its container if the container had been scrolled. Therefore, a flag, **kSFSflagPosnScroll,** has been added to control the positioning of subforms.

When the kSFSflagPosnScroll is set the subform in a subform set (SFS) will open relative to the current scroll position of its container. For example, on a long form which is currently scrolled to the bottom of the page, with a subform opening at left position 100 and top position 100, it will open 100 pixels in from the top of what can currently be seen in the viewport. Similar behavior would apply if it belongs to a paged pane that has been scrolled.

When the flag is not set, it will be positioned absolutely to the defined position: therefore, in a long form that has been scrolled to the bottom of the page, the subform will be placed at the top of the page if its top position is set to 0.

However, when opening a modal subform in a subform set, if its position is set to kSFSCenter, it will always be positioned relative to the current scroll position of the form, as modal subforms always belong to the form, not a paged pane (since a modal subform requires interaction and closing before any other action can be taken on the form). If kSFSCenter and kSFSflagPosnScroll are both not used, then a subform will be placed at its specified position, even if that is out of the current view of the user (which is the behavior in previous versions).

Maximize Open flag

A new subform set flag has been added, **kSFSflagOpenMax,** which maximises subforms within the subform set upon opening them. Sizes/positions should still be set as the subform will return to these values if it is restored.

Scrollable flag

The **kSFSflagScrollable** flag has been added to allow subform sets to scroll, when used with the ‘subformset_add' client command. The new flag only affects non-responsive subforms, since responsive subform sets are scrollable by default.

Monitor Wizard

The **Monitor** remote task wizard can now produce a remote form for displaying the connection results and activity (only a desktop window was available in previous versions): you can choose either or both when you step through the wizard.

In addition, an extra pane allows you to identify the remote task in your library that needs to have the Monitor set for its superclass; this had to be set manually in previous versions.

Serverless client methods

The default execution type for new methods added to a serverless client remote form is now client-executed.

Error Text

In previous versions, white space assigned to $errortext removed the error div, however this is no longer a problem.

Autocomplete

The default setting for the input element in the Edit control's 'autocomplete' attribute was "off", but this cleared its contents when suspending to the browser's cache. This is no longer set to "off" by default, but you can set this to a valid value.

JavaScript Components

Video Control

The JS Video control has been rewritten to remove its reliance on jQuery, and as a consequence the control has some new properties and events. The $flowplayerurl and $flowplayerline properties have been removed, as all supported browsers now support HTML5 video. In converted libraries, the updated JS Video control will continue to work as before.

The following new properties have been added:

- **$startposition**
  The time (in seconds) at which the video should start when played.
- **$currentposition**
  The current time (in seconds) of the current position in the video. Assign to this to seek to a particular time.
- **$duration**
  The duration of the current video (in seconds). Read-only (in a client-exec method).
- **$poster**
  A URL to an image to display before the first frame of the video is ready. HTML5 video only ($youtube=kFalse)
- **$playing**
  Whether the video is currently playing. Assign to this in order to play or pause the

video. Note that many mobile devices prevent the playing of videos if not in direct response to a user action.

- **$volume**
  The volume level of the video player (0-100). Assigning 0 will mute the player.
- **$playbackrate**
  The video's playback speed, with 1.0 being default speed. Youtube will round down to the closest supported rate of the particular video.
- **$requestcaptions**
  If true, closed captions will be turned on (when available, attempting to use the client's language) for Youtube videos. Note that even if disabled, captions may be enabled through the video controls, or through the user's account settings in Youtube (if they are signed in).

Properties relating to the current video player ($currentposition, $duration, $volume) will return -1 if queried before a video is 'ready' (see evVideoReady).

Events

The JS Video control has some new events.

- **evVideoReady**
  Sent when the video is ready to be played, and can be interacted with.
- **evVideoEnded**
  Sent when the video has finished playing (i.e. it has played to the end).

Both events receive a pVideoURL parameter, describing the currently playing video. For HTML5 videos ($youtube = kFalse), this will be a URL to the video file. For Youtube videos ($youtube = kTrue), this will be the Youtube video ID. These should correspond to a value in the list assigned to the control's $dataname.

Youtube Playlists

If the list assigned to a Youtube video control ($youtube = ktrue) has more than one line, a playlist will be created, using the video IDs supplied in each line of the list. The videos in the playlist will be played successively.

If $showcontrols is true, the playlist can be accessed via the video controls in the UI. You could notationally skip to the next video by skipping to the end of the current video. For example, using the client-executed method:

```
Calculate $cinst.$objs.youtubeVideo.$currentposition as
   $cinst.$objs.youtubeVideo.$duration
```

Data Grid

Column Justification

The JavaScript Data Grid has the following new properties to allow you to justify content in grid column headers.

- **$headerjst**
  A kJSDataGridJst… constant that sets the alignment of the data grid header
- **$columnheadersjst**
  A kJSDataGridJst… constant that sets the alignment of all the column headers; overrides $columnheaderjst
- **$columnheaderjst**
  A kJSDataGridJst… constant that sets the alignment of all the current column's header; $columnheadersjst must be set to kJSDataGridJstDefault

The JS Data Grid control now only escapes HTML in Character cells when formatting the oontent for display. Therefore, when you edit a cell containing Character content, you will see the same content as when not in edit mode.

Highlighting Cells

The properties $hilitefocusedcell and $cellhilitecolor have been added to the JS Data Grid to allow you to highlight the cell that has the focus.

- **$hilitefocusedcell**
  If true, the focused cell will be outlined in the color specified by $cellhilitecolor
- **$cellhilitecolor**
  The color of the focused cell's outline, provided $hilitefocusedcell is kTrue

Initial Row Values

When a Data grid has $enterable & $extendable enabled, the user can add a new row by entering data into the empty 'extendable' row at the bottom, and the remainder of the columns in that row are given default values.

However, if you want to override these defaults, you can now implement a method named $initextendrow on the Data Grid control. This method should return a row with column values set to the appropriate default values you wish to use. The order and the data type of the columns must match the order and types of the columns of the list defining the Data Grid and specified in $dataname.

Row Styles

The new $rowcsscol property allows you to specify CSS styles for a row in a data grid. The $rowcsscol property specifies the column number in the $dataname list for specifying custom CSS class names to apply to individual rows. Multiple class names can be assigned with a space separated list.

The CSS rules for classes can be added to user.css: it may be necessary to use !important to override existing styles. For example, in user.css:

```
.omnis-datagrid .highlight {
    background: red !important;
    color: white !important;
}
```

Entering Dates Manually

The properties $editdatetext and $columnallownulldateinput have been added to the Data Grid to allow end users to enter a date manually via the keyboard rather than having to use the date picker.

When set to true, $editdatetext (and $columneditdatetext when $userdefined=kTrue), allows keyboard entry of a date/time. If a date that cannot be parsed is entered, it will revert to the previously stored date, unless $columnallownulldateinput=kTrue, in which case the field data will become null.

Note this has no effect on the date picker popup control, so if you don’t want to use the picker you need to apply the following css rule to hide the picker:

```
.datetimepopup-button {
   visibility: hidden;
}
```

Toolbar Control

Selected Line Color

The behavior for the selected line in the side menu of the Toolbar control has been improved with the addition of the property $selectedlinecolor.

- **$selectedlinecolor**
  The color used for the background of the selected line in the side menu.

Items in the side menu can now have a 'selected' state as well as a 'focused' state. Selecting a line in the side menu now sets the current line in the list. The selected line

will remain highlighted until another line is selected. When the side menu is opened, the selected line will get the focus.

Note that icons are not displayed on overflow items.

Side menu & Hover Text Color

Several text color properties have been added to the JavaScript Toolbar to allow you to set the text color and hover color of items in the side menu and overflow menu. The new properties are:

- **$toolbarhovertextcolor**
  The text color of toolbar items when hovered
- **$sidemenutextcolor**
  The text color of side menu items
- **$sidemenuhovertextcolor**
  The text color of side menu items when hovered
- **$overflowtextcolor**
  The text color of overflow menu items
- **$overflowhovertextcolor**
  The text color of overflow menu items when hovered

The new properties have the value kDefaultColor by default so that any side menu and overflow menu items in a toolbar control in an existing library should have the same colors as before.

Disabling Items

The $itemenabled property has been added to JS Toolbar items to allow you to disable specific items. When $itemenabled is set to kFalse for an item it is greyed and cannot be selected with the pointer or keyboard. This property applies to items whether they are on the toolbar itself or the overflow menu.

Date Picker

The appearance and layout of the Date Picker in date/time and calendar mode has been improved. You can specify a custom format using the new property $datestylecustom and you can allow end users to select a date range using $rangeselection.

Custom Date Style

A new property, $datestylecustom, has been added to the Date Picker control, which is used in conjunction with setting $datestyle to the new kJSDatePickerStyleCustom setting. You can enter a string of characters to represent the columns required as per the Omnis date/time format strings, for example, "mdy" to specify Month, Day, Year columns in that order.

In addition, you can specify a grouped column by enclosing the date characters in parenthesis, for example, "(wdm)" will specify a single coumn containing Weekday, Day, Month. Note: this column will always alter the day by one by increasing or decreasing it, so it only makes sense to use this type of column if it includes a day or weekday. Time elements entered into a grouped column will be ignored. Repeated characters are ignored and only one group can be used (further groups are ignored). Groups take precedence over individual columns, therefore "d(wdm)y" will be treated as "(wdm)y".

Date pickers (other than custom) now pick up the locale of the client and display the picker in their standard format. For example, the Date Picker will display Day, Month, Year in the UK, and Month, Day, Year in the USA (assuming their location settings are set correctly).

These changes have also been implemented in the Data Grid. The data grid uses the appropriate Date Picker according to the constant specified in $dateformat or $columndateformat. If this is set to kJSFormatCustom, then $dateformatcustom or

$customdateformatcustom is used as above. If set to kJSFormatNone, then it will attempt to use the data subtype applied to the dataname of the column to determine which picker to use.

evDateClick event

A new event evDateClick has been added to the Date Picker, which is generated when a date is clicked, regardless of whether or not the date has changed.

Selecting a Date Range

Two new properties have been added to the calendar type Date Picker $rangeselection and $rangeenddataname to allow the end user to select a date range, that is, a *start date* and an *end date.* The first being a boolean to put the calendar into range selection mode. When true, the end user can select a range of dates by selecting one date after another. The $rangeenddataname property is the name of an instance variable to store the end of the data range and should be of type Date. The variable in $dataname will always hold the start date in range selection mode.

A boolean parameter, pInRangeSelection, has been added to evDateClick which only applies to calendar type Date Pickers. This will be passed as true when the end user has selected the first date, and false once they have selected the second. If $rangeselection is kFalse, this parameter is not passed, and therefore will return NULL if tested on evDateClick.

A new event, evDateRangeChange, has been added, which fires every time a date range selection has been completed (and $rangeselection is kTrue). This passes two parameters: pStartDate and pEndDate. This means you can obtain a date range without using instance variables if you just need to react to the date range selected. evDateChange does not fire when $rangeselection is kTrue.

As part of these changes, $currdaycolor now applies to inside the current day indicator ring instead of applying to the whole cell. This ensures the type of cell is still understood by the end user. E.g. When $todayscolor is different to $daycolor, the end user can still see that it is today, even when they have selected it as the current day.

Localization

The following strings have been added to the JS Localization string table to allow you to localize strings for the Date Picker, and some generic labels for other controls. Note that some of the strings are now arrays of strings to simplify localization (e.g. for months, days of the week, etc).

Date picker specific strings

The following are specific to the Date Picker control:

```
"ctrl_date_increase": "Increase"
"ctrl_date_decrease": "Decrease,
"ctrl_date_time_button": "Open time picker"
"ctrl_date_calendar_button": "Open date picker"
"ctrl_date_header": ["Select a Month", "Select a Year", "Select a Decade",
   "Select a Time"]
```

Generic strings

The following are more generic strings for months of the year that may be used across different controls:

```
"month_names": ["January", "February", "March", "April", "May", "June",
   "July", "August",
"September", "October", "November", "December"]
"month_names_short": ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
   "Sep", "Oct",
"Nov", "Dec"]
"day_names": ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
   "Friday",
"Saturday"]
```

```
"day_names_short": ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
"date_units": ["Day", "Month", "Year", "Decade"]
"time_units": ["Hour", "Minute", "Second", "Millisecond"]
```

The following example applies Spanish text to the Calendar:

```
<script type="text/javascript">
  jOmnisStrings.es = {
   "month_names":   ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
   "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"],
   "month_names_short":  ["enero", "feb.", "marzo", "abr.", "mayo", "jun.",
   "jul.", "agosto", "sept.", "oct.", "nov.", "dic."],
   "day_names":     ["Domingo", "Lunes", "Martes", "Miércoles", "Jueves",
   "Viernes", "Sábado"],
   "day_names_short":    ["DOM", "LUN", "MAR", "MIÉ", "JUE", "VIE", "SÁB"],
   "date_units": ["Día", "Mes", "Año", "Década"],
   "time_units":       ["Horas", "Minutos", "Segundos"],
   "ctrl_date_header":   ["Selecciona un mes", "Seleccione un año", "Seleccione
   un década", "Seleccione una hora"]
};
</script>
```

See the Localization chapter in the *Creaing Web & Mobile Apps* manual for more information about setting the strings in the jOmnisStrings object.

Tree List

Line Border

The JS Tree List has two new properties, $lineborder and $linebordercolor, that allow you to add a horizontal line between nodes. When $lineborder is set to true, a row border is added between each node. $linebordercolor specifies the color of the line; it uses the value of $bordercolor when set to kColorDefault.

In addition, when selecting a node in a Tree List the whole line is now selected. This makes it more consistent with the appearance of the thick client, and also the Windows and macOS native behavior.

Even Row Color

A new property, $evenrowcolor, has been added to the Tree List and specifies the color to be used for every even row in the list of nodes. The kColorDefault setting means use the same color as odd numbered rows ($backcolor). This can be used to produce a similar effect to the macOS finder, with alternating colors on odd and even rows.

Complex Grid

Drop support has been added to the JS Complex Grid which allows the end user to drag data from a remote form field and drop it onto a cell in a complex grid. To allow drop support, the events evCanDrop and evDrop, and the $dropmode property have been added to the Complex Grid.

The pDropRow event parameter is available for evCanDrop, evWillDrop and evDrop events, and reports the row of the complex grid on which the drop is to occur (zero if the control does not belong to a complex grid).

It is possible to drop data onto a single control in the grid (a cell) in any row in the grid, as long as it has its $dropmode enabled. If not, the complex grid itself will receive the drop: this differs from the fat client complex grid in which only fields on the current row can receive a drop.

List Control

Double-click Events

A double-click event is now sent to a JS List Control if the *Enter* or *Space* key is pressed while the focus is in the List *and* if the evDoubleClick event is enabled on the list. Otherwise, if the control has an evClick event enabled, Enter/Space sends an evClick. If the control does not have the evClick or evDoubleClick enabled, then Enter triggers the okkeyobject (if there is one), as long as the state of the list has not changed, i.e. the current line has not changed, and no checkbox has been toggled.

List Pager

The $pagerpage property has been added to all JavaScript controls that support the List pager to allow you to set the page to be displayed; $pagesize must be greater than zero for this property (and the List pager) to be enabled. The List pager is supported in the standard JS List, Native list, Data grid, and Complex grid.

Edit Controls

Accented Characters (macOS)

The mechanism on macOS for entering accented characters using the Option key now works for all built-in Edit fields. Note there was a fault stopping this working when dictation was enabled but this is now fixed.

For example, on a British keyboard typing Option-N provides the dead key for the ~ accent and will display that as the selected character. The next key stroke determines the replacement character to be accented, so typing n will replace the ~ with ñ.

Auto Correction (macOS)

You should note that Auto correction, Auto capitalization and Auto completion (the properties $autocorrect, $autocapitalize & $autocomplete) only work in Omnis when they are enabled on the client Mac computer. Note also that $autocapitalize only applies when using a virtual keyboard on a device.

Selecting Dates

A new constant, kJSInputTypeDate, has been added to the $inputtype property of the Edit Control to allow the end user to select a date using the Date Picker. When $inputtype is set to kJSInputTypeDate (and $inputtypetouchonly is set to false), a date/time picker will be used to pick a date value for the Edit control. $dataname must be set when using kJSInputTypeDate and other input types such as kJSInputTypeNumber.

The format of the date picker should be calculated from $dateformat ($dateformatcustom if $dateformat == kJSFormatCustom). If $dateformat is kJSFormatNone, then the control attempts to fall back to the dataname subtype.

Paged Panes

Paged panes now scroll during design mode when you drag and drop content onto the page control. The drag and drop scroll rectangle has changed and now includes the edge of the client when scrolling with the mouse at the right or bottom edge of the container.

Switch Control

The JS Switch Control has two new properties, $justifyhoriz and $justifyvert, to justify its contents horizontally or vertically. $justifyhoriz can be set to kLeftJst, kRightJst, or kCenterJst, while $justifyvert can be kJstVertTop, kJstVertBottom, or kJstVertMiddle.

TransButton

The JS TransButton will now center its content vertically when $vertical is true: it previously anchored text/icon to top/bottom of the control when $vertical was true.

$align now also affects the placement of the icon when $vertical is true.

Disabled Appearance Property

A new property, $defaultdisabledappearance, has been added to all JavaScript controls which have the $enabled property. The property defaults to true, which, when the control has $enabled = kFalse, applies the 'omnis-notenabled' css class to the client element. If $defaultdisabledappearance = kFalse, this class is not applied, which is what sets the text colour to grey when disabled.

The controls which have this new property are: Bar Chart, Combo Box, Data Grid, Date Picker, Edit, List, Map, Pie Chart, Rich Text, and Tree list.

Accessibility Properties

It is possible to use a space separated list of controls for the $arialabelledby and $ariadescribedby properties to assign multiple controls as labels for the component.

Component Store

The bitmap for an object dragged from the Component Store now has a rectangle with the same dimensions as the control that it will drop.

Field List

The Field List (which lists controls on a remote form, as well as windows and reports) now scrolls to the first selected field to display it, expanding tree nodes if necessary.

Commands

Line: command

There is a new command, *Line:,* which is like the *Text:* command, except that it just adds a single line of text to the text block.

Syntax

Line: *line-text*

Description

Adds a line of text to the text buffer for the current method stack. The *Line:* command supports leading and trailing spaces and can contain square bracket notation, that is, you can include or add the contents of a variable to the text buffer. You build up the text block using the *Begin text block* and any combination of one or more *Text:* or *Line:* commands. The Carriage return and Linefeed options of the *Begin text block* command specify the line delimiter added to the text buffer after the text added by the *Line:* command. When you have placed one *Line:* command and you press Ctrl/Cmnd-N to create a new method line, a new *Line:* command is added. You should end a block of text with the *End text block* command, and you can return the contents of the text buffer using the *Get text block* command.

There is a new external editor (similar to the JavaScript: and Sta: editors) for adding consecutive sequences of Line: commands. You can open this in the usual way, via the code editor parameter helper. Note that Line: is available in both normal and client-executed methods.

Begin text block command

The *Begin text block* command has two new options, Carriage return and Linefeed. These specify the line delimiter added after each line of text added by the *Line:* command. If you omit both of these options, Omnis uses the platform specific newline character.

Window Classes & Components

OBrowser

CEF support on macOS

The macOS version of OBrowser (window class component) now uses the Chromium Embedded Framework (CEF), which is already used in the Windows version of the component.

With this enhancement, the macOS version of OBrowser now supports the standard OBrowser CEF configuration settings using the *cefSwitches* configuration item within the config.json (same as the Windows version).

The previous version of OBrowser on macOS (which used the Cocoa WebView) had the property $disablepluginsmacos, but this is now obsolete and will not show in the Property Manager. The notation for this property is still supported in your code but it has no effect.

Cookies

The Chromium Embedded Framework (CEF) used by OBrowser stores cookies in a SQLite database called Cookies. This is located in the user App Data folder, such as on Windows:

```
C:\Users\<username>\AppData\Local\Omnis Software\OS10.x\chromiumembedded\cache
```

Or in the /Application Support folder at /chromiumembedded/cache on macOS. This database can be managed using a SQLite DAM session.

Object Animation

There is a new library property, $animateui, that controls whether or not certain window class controls are animated; those same controls also have the $animateui property. For this release, the **Tree List** has the new property, and the **Tab Strip** has some new types to highlight and animate the tabs when they are selected (animations will be applied to more controls in future releases). The property is defined as:

- **$animateui**
  If the library property $animateui is true, all objects that support $animateui will animate aspects of their interface. Therefore, the object property only applies when the library property is false.

If the $animateui library property is false (shown on the Appearance tab in the Property Manager), the setting of the $animateui property for the individual object is used. Therefore, if you only want *some of the controls* in your library to animate, set the $animateui library property to false, and override at the object level by setting $animateui for the object to kTrue.

Tree List

When $animateui is enabled for a Tree List, the contents of the list will display by dropping down gradually as you open the control. If $animateui is disabled the tree list content drops down instantly, as in previous versions.

Tab Strip

The $squaremode property in the Tab Strip has been enhanced and now includes several different settings to provide new appearance and animation options (the $animateui library property must be enabled to allow the animation). The $squaremode property is set to kTabStripOriginal by default which means it has the same appearance and behavior as in previous versions. The other options include:

- **kTabStripSquare**
  the tabs have square corners and fill the entire control area; there is no animation when the tab changes
- **kTabStripAnimSquare**
  the tabs have square corners and the tab change is animated
- **kTabStripAnimLine**
  the current tab is indicated with a *line* and the tab change is animated
- **kTabStripAnimDot**
  the current tab is indicated with a *dot* and the tab change is animated
- **kTabStripAnimRndSquare**
  the tabs have rounded square corners and the tab change is animated

The $animateui library property *must be enabled* to use the Tab Strip animations; if the preference is set to false, you can still use the square, line and dot options but they will not be animated.

IDE Animation

As a consequence of adding animation, some controls in the Omnis Studio IDE are animated. For example, the main tree list in the Studio Browser, and the Method names tree list in the Method Editor is animated when you open the editor or redraw the list.

There is a new option "animateIDEcontrols" in the "ide" section of config.json that enables animation in the IDE (this does not affect the setting of the library or object property in your own libraries, just the IDE): it is set to True by default. Set this to false if you don't want any objects in the IDE to be animated.

Animation Curves

The $beginanimations() method now has an extra parameter allowing you to specify one of a number of new animation "easing" curves. The definition for $beginanimations() is now:

- **$beginanimations**(iDuration[,iCurve=kAnimationCurveEaseInOut])
  after calling this, assignments to some properties are animated by $commitanimations() for iDuration (in milliseconds), using iCurve as the animation curve (kAnimationCurveEaseInOut is the default)

Where iCurve can be one of the following animation curves:

- **kAnimationCurveEaseIn**
  The animation begins slowly and then speeds up as it progresses.
- **kAnimationCurveEaseInBack**
  The animation is similar to kAnimationCurveEaseIn but first moves in the opposite direction before easing begins.
- **kAnimationCurveEaseInOut**
  The animation begins slowly, accelerates through the middle of its duration, and then slows again before completing.
- **kAnimationCurveEaseOut**
  The animation begins quickly and then slows down as it progresses.
- **kAnimationCurveEaseOutBack**
  The animation is similar to kAnimationCurveEaseOut but moves beyond the final point before easing back to the final location.
- **kAnimationCurveEaseOutBounce**
  The animation starts slowly and then bounces on its final location.
- **kAnimationCurveEaseOutElastic**
  The animation starts fast and springs to a stop around its final location.
- **kAnimationCurveLinear**
  The animation occurs evenly over its duration.

Switch Control

There is a new **Switch Control** that has the appearance of an “iOS style” switch: when the switch is turned on or off (clicked) the round button slides across and the background changes color. The on/off state is assigned to the $switchon property. The following shows the off (left) and on state:

The Switch Control is an External Component and will appear under the External Components tab in the Component Store, but it is not loaded by default. To load the Switch control, right-click on the Component Store, select External Components, open the External Components group, scroll and select the ‘Switch Library’ in the list, and finally select the Pre-Load status (on opening Omnis or the current library). The Switch control will now be visible in the Component Store and can be dragged onto your window: note you will need to resize the control to make the background part visible.

The Switch Control has the following properties (shown on the Custom tab in the Property Manager):

- **$switchbutton**
  the color of the round button part of the switch; the default color is white
- **$switchcolor**
  the background color of the switch when switched on; the default color is dark green
- **$switchon**
  true if the switch is on; setting this in design mode sets the default state when opening the window
- **$transparencywhenoff**
  the amount of transparency when the switch is turned off, an alpha value from 0 to 255; the default value is 50

You can test the value of $switchon in the $event method for the control to branch depending on its true/false value.

Multibutton Control

There is a new **Multibutton** control that provides a round, animated popout button that opens to show a number of additional options, each represented by an icon. The button reports the evButtonClicked event with the pButtonid parameter being the selected button. The following image shows the closed state (left) and open state of a multibutton, in this case opening to the right:

The Multibutton Control is an External Component and will appear under the External Components tab in the Component Store, but it is not loaded by default. To load the Multibutton control, right-click on the Component Store, select External Components, open the External Components group, scroll and select the ‘Multibutton Library’ in the list, and finally select the Pre-Load status (on opening Omnis or the current library). The Multibutton control will now be visible in the Component Store and can be dragged onto your window.

The Multibutton Control has the following properties (shown on the Custom tab in the Property Manager):

- **$buttoncolor**
  The background color of the control

- **$buttonopen**
  kTrue if the control is open
- **$expanddirection**
  the direction the control expands, a constant: kMBexpandRight, kMBexpandLeft, or kMBexpandCenter
- **$iconstr**
  comma separated list of icon ids that are displayed when the control is opened, the number of icons determines the number of options; you can provide icons with a transparent background, so the background color is seen
- **$openicon**
  the ID of the icon shown to 'open' the control; this is shown in the closed state and can be a different icon as those displayed in the popped out list of buttons
- **$closeicon**
  the ID of the icon shown to 'close' the control; this replaces the icon specified in $openicon

The multibuton reports the **evButtonClicked** so you can use this in the $event method for the control and test the value of pButtonID which is the id of the selected button starting at 1 for the first button in the popped out list of buttons.

Window Messages

The $showmessage() method has been added to window instances, which you can use as an alternative to the OK message command; this is similar to the $showmessage() method which is available in remote form and remote task instances. The new method is also available for menu, toolbar, report, object, and table instances. The method has the following definition:

- **$showmessage**(cMessage[,cTitle,iOptions=kMsgOK])
  displays a message using the specified cMessage, cTitle and iOptions (a sum of kMsg... constants). Returns true for OK or Yes, false for No or cancel. You can use msgcancelled() to check for cancel.

The supported constant values are:

kMsgOK	Display an OK message (the default)
kMsgYesNo	Display a Yes/No message
kMsgNoYes	Display a No/Yes message
kMsgCancelButton	Add a cancel button to the message
kMsgIcon	Display an operating system specific icon with the message
kMsgSoundBell	Sound the bell when the message is displayed

If you mix kMsgYesNo, kMsgNoYes and kMsgOK, kMsgYesNo has precedence over kMsgNoYes. kMsgNoYes has precedence over kMsgOK.

Window Design Task

When testing a window class (Ctrl/Cmnd+T or the Open window hyperlink of the browser, or the browser context menu for a window class) Omnis now switches to an instance of the design task, or the startup task if there is no design task.

There is a new item in the config.json, tryDesignTaskWhenTestingWindow in the "ide" section, to control the new behavior. When true (the default), Omnis looks at the design task name, and if it is the same as the startup task name, switches to the startup task, as before. If however, the design task name is different, Omnis switches to the first

instance it can find of that task, but if there is none, it switches to the startup task as before.

When tryDesignTaskWhenTestingWindow is false, the behavior is the same as for previous versions: when testing a window, Omnis switches to the startup task of the library containing the window.

List box, Headed List and Check box lists

The $linebackgroundcol property has been added to the List box, Headed List and Check box list windows class controls.

The $linebackgroundcol property specifies the column number in the data list for the control ($dataname) that contains color values that override the default background color of each line; the value zero or kColorDefault in this column means the normal background color for the line is used.

Font Scaling for Fields

You can now increase or decrease the font size of fat client Multi-line Entry field, String grid and Data grid using the key press Ctrl + or Ctrl -.

The $disablefontsizekeys property has been added to control font scaling for these controls, together with the standard List, Checkbox list, Headed list and Tree list which already respond to the Ctrl +/- key press to scale the font. The default value of $disablefontsizekeys is kFalse, which means the control will respond to the Ctrl +/- key press to adjust its font size; set the property to kTrue to disable font scaling.

Headed List

You can now use the style() function to style the text in Headed List box column names specified in $columnnames. Note the styled text in $columnnames has to be assigned in $columnnames at runtime.

Round Button

The Round button control, introduced in Studio 10.0, uses transparency, so requires a minimum of Windows 8 or higher.

Field Styles

The style for a window class object is stored in its $fieldstyle appearance property. When adding a property in the custom style dialog (opened from a context menu in the Property Manager) the current style (if any) is selected in the dialog by default.

Background Object Names

Windows class Background objects, such as Label and Text objects, now have the $name property which defaults to the ident of the object, that is, a number which is generated automatically (e.g. 1016). You can assign your own name which will help you identify the object more easily: you cannot set $name to a number, but the name can include numeric characters. Setting the name to empty resets it to its default number ident.

Border Effects for Shape, Text and Labels

Shape fields, as well as Label and Text background objects now support the $effect, $bordercolor and $linestyle properties to allow you to apply border style effects.

Functions

delchars()

There is a new function delchars() which can delete a substring of a specified length at the specified position.

delchars(string,position[,length=1])
Returns the string generated by deleting the substring at the specified 1-based *position* and *length* from *string*. If length extends past the end of the string, the function deletes to the end. For example:

```
Calculate lString as delchars('String',2,2)
#  lString is now 'Sing'
```

sys(123)

There is a new sys() function, sys(123), that returns the build number of the Omnis executable. The Omnis About box shows the same build number in the version string.

sys(192)

sys(192) now contains the line number of *the executing line,* and the *executing linetext,* rather than the next linetext to execute.

A Boolean item has been added to the config.json file, "sys192excludesIDEmethods" in the "defaults" section, to specify whether or not to include IDE method calls in the list returned from sys(192) (the default is true so IDE methods are excluded). When true, this will exclude an IDE method if the library containing the method is marked as always private.

sys(292)

There is a new sys() function, sys(292), that returns an empty or single line list containing the same columns as sys(192) where the line represents the calling method.

systemversion()

The systemversion() function has been added to replace all the "is...()" functions (e.g. iswindows10()) for determining the current operating system version.

Returns a row of version information about the current operating system, with columns platform, major, minor, build, and server.

The platform parameter is the platform code; major, minor, and build identify the system version (not for Linux); server is true for Window server systems.

pictformat()

pictformat() is now available in the Linux headless server.

isclear()

The isclear() function now returns true for *empty* and *false* Boolean values.

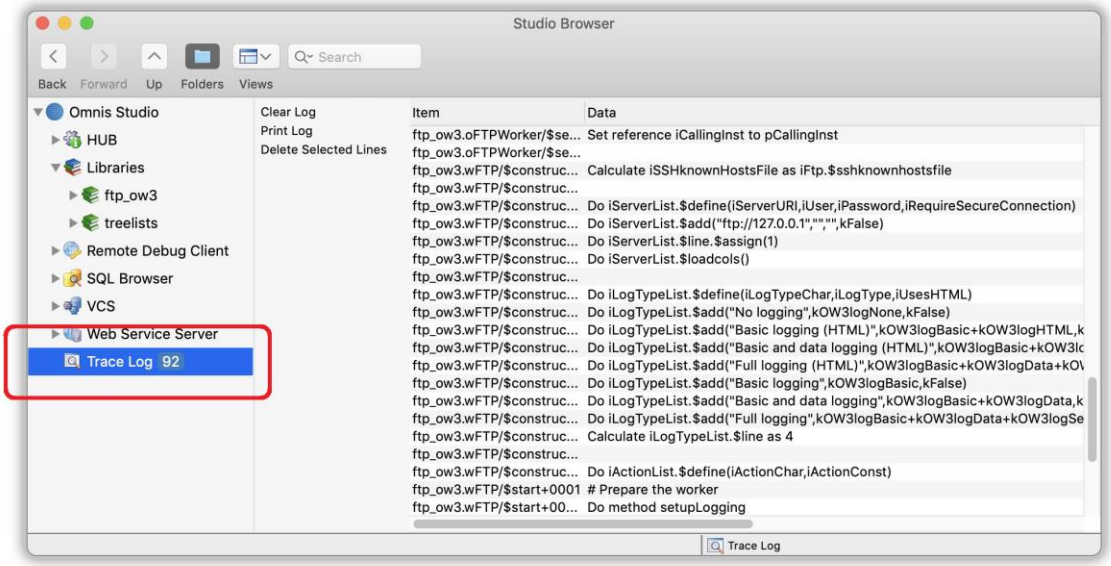
FileOps.$splitpathname

The last three parameters (directory-name, file-name, file-extension) of FileOps.$splitpathname are now optional.

Omnis Environment

Trace Log

The Trace Log has been added to the Studio browser. There is a new node in the Studio browser to open an alternative view of the Trace Log, which behaves in the same way as the existing trace log (except there is no max lines setting). The Trace Log node shows the current number of lines in the log.



There is a new option in the HUB options to specify whether or not the Trace Log node is displayed in the Studio Browser: the default is on.

The new Trace Log view in the Studio Browser provides all of the options currently accessible via the trace log window (either via hyperlinks, context menu or both), with the exception of the maximum number of lines setting (which now defaults to 100000). Also, double clicking on the trace log view behaves just like the trace log window.

You can use the search box at the top of the Studio Browser to search the contents of the trace log.

The current trace log window is still available, in the Open trace log command, and also for the runtime.

Server Socket Bind Failures

There is a new option to disable Trace Log opening in runtime when a server socket bind fails. In the developer version, the Trace Log window now never opens when this occurs as you can view the trace log in the browser.

The new option is a Boolean property in the ‘server’ section of config.json: "runtimeOpensTraceLogOnSocketBindError"; the default value is true, so set this to false to suppress the trace log.

Update Manifest Files (macOS)

On macOS if an Omnis deployment is replacing an existing installation using a simple drag and drop approach, that is, from a disk image to the Applications folder, files which already exist in the current user's Application Support folder will not be updated from the files in the firstruninstall folder of the new disk image.

If there are files which need to be patched, a mechanism needs to be found to remove the existing files from the user data location, e.g. run an update script. Therefore, Omnis now provides the use of *Update Manifest Files* to allow a deployment to specify

the files in an existing set of user data which need to be removed so they can be replaced by newer files from the firstruninstall folder.

When Omnis starts it will read an integer deployment version number from a file called "version" in the Omnis application's macOS folder:

```
/Applications/Omnis\ Studio\ 10.1.app/Contents/MacOS/version
```

If this file does not exist then the deployment version number will be set to the Omnis internal build number (as returned by sys(123)).

Omnis will read the version of the user data from a hidden file (.version) in the root of the user's application data folder for the Omnis application.

```
/Users/<username>/Library/Application Support/Omnis/Omnis Studio 10.1/.version
```

If this file does not exist, the user data version is set to zero. If the user data version is lower than the deployment version, Omnis will check for updates that need to be applied.

The updates are specified in a set of files which should be placed in a folder called "manifest" within the Omnis application's macOS folder. Each file should be named for the version which specifies the changes. For example, if the new deployment is version 23071 there should be a file named 23071.

```
/Applications/Omnis\ Studio\ 10.1.app/Contents/MacOS/manifest/23071
```

The manifest file should contain the paths of each file or folder which needs to be updated for that version. Each path should be separated by a new line.

Therefore, if file 23071 contains:

```
studio/v40.lbs
startup/vcs.lbs
```

This will indicate that the Studio Browser and VCS are to be updated (removed from the user data) for version 23071.

There can be a manifest file for interim versions if updating an older version of Omnis. Therefore, if the user version is 23069, the deployment version is 23071 and the manifest folder contains 23070 and 23071 then both files will be used for updating.

If updates are applied, the hidden user data version is updated to the deployment version.

Note that the Omnis application deployment tree (code-side) still needs to be re-signed (and notarised) for each version of a deployment.

It is not recommended that individual files are updated in an existing signed tree (as this will invalidate the signature).

Query builder

You can now set the default Join type for queries in the Query Builder. The 'Join Type' selection has been added to the Joins window, which can be opened by right-clicking on a Join and selecting 'Joins' in the query window.

Appearance Property

A new color 'colorheadertextwin' has been added to 'header' section of the 'appearance.json' file to set the default text color for Headed list buttons on the Windows platform.

Libraries and Classes

Library Conversion

Conversion Messages

You can disable the working messages such as "Converting class..." during library conversion by setting a new option "showLibraryConversionWorkingMessage" located in the "defaults" section of the config.json file. The new option defaults to true, but you can set it to false to disable the conversion working messages.

Conversion Log Delimter

The log file created on library conversion (and written to logs/conversion folder) now uses tab-delimited format, with exported text in quotes (the default). You can change both of these options using new configuration items in the config.json file, in the log section:

```
"conversionLogDelimiter": "\t",
"encloseConversionLogTextInQuotes": true
```

If conversionLogDelimiter is empty, Omnis uses the default log delimiter, a semicolon (;).

Error Processing

There has been a small update to $clib.$prefs.$errorprocessing logging. The log messages have been reduced to a single line identifying the error, and the call stack lines.

Default Library name

The characters and format of the string allowed in the $defaultname property (to override the auto-generated internal library name) for a library are now more limited. When assigning the library $defaultname, you cannot include leading or trailing spaces, and the name cannot start with a digit or contain the following characters: . $ ( ) [or].

$container

The $container notation has been extended to object and table instances.

You can use $cinst.$container in an object or table instance, to reference the instance that contains the object list or row variable. This may return #NULL if there is no associated instance, such as, when the variable is a class variable.

JavaScript Worker

List/Row Parameter

In the JavaScript worker, the list/row parameter passed to $methodreturn now has two columns added by the worker: __module and __method. If the parameter is a list, then __module and __method are only populated for the first line of the list.

The list/row parameter of JavaScript worker $callmethod method is now optional. An empty parameter is passed to the JavaScript method as null.

Non-JSON content

The content column for non-JSON content returned from a JavaScript worker method is now of type character when the content type is text/.

Node.JS Error Reporting

Reporting of node.js errors has been improved. If an unhandled exception occurs causing node.js to exit, Omnis now adds the stack traceback of the exception to the errorInfo column of the row passed to $workererror.

Remote Debugger

Locked Classes

You can now debug methods in a class when it is locked using the remote debugger if the new class property $canremotedebugwhenlocked is set to true. Locked classes for which this property is kFalse do not appear in the remote debug client list of classes for the library.

Exclude Folders

A new option has been added to exclude folders from Remote Debugger class list. The Remote Debug Server configuration has a new option (Exclude folders) which controls whether or not folders are returned by the server to the client, and therefore displayed in the browser. The remote debug server dialog has been updated to allow this option to be edited.

Omnis Datafile Migration

SQLite logon configuration file

You can now specify a logon configuration file when connecting to a SQLite database file (this was previously only available for PostgreSQL).

The logon configuration file for SQLite should have the .DFQ file extension, and can be specified as the hostname when opening the database. The DFQ file may contain one or more session property assignments, e.g. quotedidentifer=1. If 'hostname' is not present the library uses the pathname of the .DFQ file and substitutes .DB.

PostgreSQL logon prompt

When emulating DML using PostgreSQL there is a prompt if either hostname, username, or password is missing from the logon config file. A "prompt" item may also be specified in the config file to override the default prompt message.

List Programming

List & Row Variable Columns

The maximum number of columns for list and row variables is now 32000, not 400 as in previous versions.

Object Classes

Object Variable Count

The $usage property has been added to object variables to report the current number of object variables that are sharing the underlying external component object. A NULL

value means the object is neither an external component object, nor is it subclassed from an external component object.

Note that copies of the object such as $statementobject and $sessionobject contribute to the count.

File Classes

Defining a List from a File class

You can now define a list based on a file class in another library using the notation list.$define(): note that the name must be passed as a string e.g. "lib.filename".

You can use the switch /s, e.g. "lib.filename/s", where s means skip columns with empty names in the file class.

Web Services

Unknown Query String Parameters

RESTful methods can now allow unknown query string parameters.

The RESTful panel for a RESTful method in the Method Editor has a new checkbox option "Allow unknown query string parameters" (the default is unchecked). When checked, it means the RESTful server will accept requests that contain query string parameters that are not specified in the method parameters. The remote task instance can access these unknown parameters using a new property $unknownquerystringparams (e.g. using the notation $cinst.$unknownquerystringparams).

The new properties are:

- **$allowunknownquerystringparams**
  If true, the RESTful method allows query string parameters that are not present in the method parameters. You access these unknown parameters using the property $unknownquerystringparams of the remote task instance.
- **$unknownquerystringparams**
  If unknown query string parameters are allowed, then this property is a row with a character column for each unknown parameter (the column name is the parameter name in lower case and the column value is the parameter value).
  There is also new notation of a method item, to reflect the new checkbox option.

The Library JSON import/export option, and method printing has also been updated to handle these new properties.

RESTful Output Type

RESTful methods can now return an array of JSON objects by suffixing their return type schema name with []. To do this, you should use schema[] as the RESTful output type, or the return type for one of the HTTP status codes, e.g. mySchema[]. The RESTful method must then return a list defined from the schema rather than a row.

Object array output type

In addition, the $sendlistsasobjectarray property has been added to RESTful HTTP methods. When set to true, the JSON generated by Omnis for a returned row or list that contains lists, contains arrays of objects rather than arrays of arrays (in this case the lists must only contain columns with simple types). There is one exception to this rule. If the list to be converted to JSON has a single column named "<array>", Omnis outputs the list as an array.

There is a new checkbox on the RESTful panel for the HTTP method in the method editor, that allows you to select this option.

Note that this option applies to both rows returned by the method, and lists returned by the method when the return type is schema[]. In the latter case, the top-level array returned is always an array of objects, therefore you should note that the new option applies to lists contained in the returned list.

Report Programming

Hyperlinks in PDF Reports

Hyperlinks are now supported in PDF and Page Preview report destinations.

To add a hyperlink to a report, use the HTML link external component (text, picture and icon are all supported), and set the $address property to the target of the link, for example:

```
https://omnis.net
mailto:bob.smith@omnis.net
```

or for Page Preview reports only, and ignored for PDF reports:

```
omnis:p1,p2,p3,p4
```

where the data after omnis: is a comma separated list of parameter values, which can be integer or character, and must not contain ".

In the latter case, when the user clicks the omnis: link, Omnis looks for a method called $previewurlclicked. Firstly, if the report has been sent to a window field, Omnis looks for the method in the window instance containing the screen report field, otherwise if the first test fails, Omnis looks for the method in the task that printed the report. If Omnis finds the method, it calls $previewurlclicked passing it the following parameters:

1. An item reference to the report instance.
2. The ident of the report object.
3. A row created by adding a column for each comma delimited item in the data after omnis: in the link.

You can create a method called $previewurlclicked in either the window or task, as above, and react to the parameters passed.

Report PDF Files & Fonts

In previous versions there was an issue printing report files to the Omnis PDF device that contain the New York legacy Mac font.

To overcome this issue, you can now map unknown macOS fonts in reports sent to PDF to alternative fonts. There is a new item called "unknownMacOSFonts" that you can add to the "pdf" section of config.json to specify the font mapping. For example:

```
"pdf": {
       "unknownMacOSFonts": {
           "New York": "Times New Roman",
           "default": "Lucida Grande"
       }
   },
```

The members of the unknownMacOSFonts object are the names of the unknown fonts to be mapped, and the name of the replacement font. A "default" member can be included to map all other fonts not listed in unknownMacOSFonts to the specified font.

Cross Platform Fonts

In previous versions, when printing a report to a binary file (via $devices.Memory), the fonts in the report were not always displayed correctly when the report files were

generated and displayed on different platforms, or if the report file was generated in the dev version of Omnis and displayed in the Runtime version of Omnis.

This issue with cross-platform fonts has now been fixed, but you should note that the fix only applies to newly created report binaries: therefore, existing report binaries with this issue will still exhibit the problem, as it is caused by font information stored in the report binary.

Using style() in Reports

In previous versions, printing a report from a remote form did not show icons retrieved using the style() function, but this has been fixed.

The style() function usually generates different results when used in a remote task instance, and the resulting style is suitable for use with the JavaScript client. The fix allows normal, non-JS client results, to be generated using style() when running in a report instance inside a remote task.

However, even with this fix, you should note that the call to print the report from the remote form, which passes the results of style() from some remote form code, will not work: you need to pass the icon id as the parameter, and call style() from within the report instance to make this work even with this fix.

Printing Background Images to PDF

When printing to PDF on a Linux headless server, Omnis now reports an error when attempting to print an object that the headless server cannot print to PDF.

Background images on reports need to be shared pictures (PNGs) to print on the Linux headless server. To solve this, go to the library prefs and set $sharedpictures to kSharedPicModeTrueColor. Then open each affected report class in the report class editor; Omnis will ask if pictures are to be converted to shared, so respond with Yes. The reports will now print to PDF in the headless server, and additionally the report classes are much smaller.

Localization

Overriding the Language

You can now override the current language set in the localisation data file by setting an item in the Omnis config.json file, which can be used with the Linux headless server. The new entry is called "language" and is located in the "defaults" section. It defaults to empty, which means the setting in omnisloc.df1 will be used. To override the language setting in omnisloc.df1, you can add the name of a language in omnisloc.df1 to the language item.

This fixes an issue when using the $formatstring property on Linux Headless Server and the German locale (LANG=de_DE.utf8).

Studio.stb file

In previous versions, you could not include cr (carriage return), lf (linefeed), and tab in strings in the studio.stb file. These characters can now be represented by <cr>, <lf> and <tab> in studio.stb. The Find Strings dialog automatically generates these escape sequences.

JSON Control Editor

JavaScript Variable Prefix

There is a new 'Options' item on the JSON Control editor toolbar to allow you to set custom JavaScript variable prefix for properties.

OJSON

$listorrowtojson()

A new iOptions parameter has been added to the $listorrowtojson() method in the OJSON external component, to allow null and empty values to be omitted from the returned JSON. The new definition for the method is:

- **$listorrowtojson()**
  $listorrowtojson(vListOrRow [,iEncoding=kUniTypeUTF8, &cErrorText, iOptions=kOJSONoptionNone]) converts the list or row to JSON. Returns JSON with specified encoding (UTF8, UTF16BE/LE, UTF32BE/LE or Char). Returns NULL and cErrorText for an error

The iOptions parameter can be used to make $listorrowtojson process all members of a top-level row, and discard empty or null values appropriately, recursively descending into child lists and rows.

The iOptions parameter can be one of the following constants, which can be summed together to get the desired result:

- **kOJSONoptionNone** (the default)
  0 - No option specified - results in the old behavior
- **kOJSONoptionOmitEmpty**
  1 - Omit empty values, objects and arrays from the output JSON
- **kOJSONoptionOmitNull**
  2 - Omit NULL values from the output JSON

OW3 Worker Objects

HASH Worker Object

Support for MD5 and HMAC hashes has been added to the OW3 HASH Worker Object. There is a new constant for MD5, kOW3hashMD5. HMACs can be generated for all hash types except PBKDF2 and the SHA3 hashes.

To generate an HMAC rather than a hash, supply the binary key as the hash parameters parameter of $inithash – an empty row as this parameter generates a hash rather than HMAC.

To verify an HMAC rather than a hash, supply the binary key as a new binary last parameter to $initverifyhash() – this is optional and its presence indicates HMAC.

FTP Worker Object

$progress can now be called for synchronous operations.

HTTP Worker

The version of libcurl and other externals used in the OW3 HTTP Worker has been updated, as follows:

- Updated curl to version 7.65.3

- Updated libssh2 to version 1.9.0
- Updated mbedTLS to version 2.16.2

Deployment

Headless Server Log Files

You can now generate a daily log for the Omnis Server by setting a new "daily" member in the "logToFile" item in config.json.

The item defaults to false, which means the current hourly logging is used. If set to true, Omnis creates a new log file for each day. In addition, Omnis re-uses the log file for a day if it is already present at startup. The rollingcount applies as for hourly logs.

Auto Update

When running the auto update script some feedback that the script is running can now be provided in a console window. To enable this, you must place a file (which can be empty) named ‘showconsole.txt’ in the same directory as update.bat. When this file is present, a console window is displayed while update.bat is running.

Omnis data folder

Resources 25599 and 25600 can now be used to specify the Omnis data folder on the Windows platform. You can edit the omnisdat.dll string table with a resource editor and modify 25599 and 25600 to be used to specify the sub-folders of the appdata directory. The Omnis data folder becomes:

```
<appdata folder>\<resource 25599>\<resource 25600>\
```

If resource 25599 is not empty, resource 25600 must also not be empty.

Omnis VCS

Project Revisions

There is a new option in the VCS, "Project Revisions", that opens a window which allows you to see all the revisions to a project, rather than having to drill down to a class and see the revisions which are only available for that class.

The Project Revisions option is available at the project level as a context menu option in the tree list or via the hyperlinks when clicking in the list of projects. You can filter the revisions from the droplists according to the user who created the revision or a date period. You can also filter by typing in the edit box.

You can drilldown to see which labels have been applied to the revision by right-clicking on it. This window also allows you to compare or copy out in the same way as the existing class level revision window. You can also select multiple classes to copy out multiple revisions. It is also possible to double-click rather than using the context menu to view the revision data.

Exclude Classes

The Omnis VCS now allows you to exclude specific classes from a build. The context menu on the project class list now includes the option "Exclude Classes" which allows you to select classes you do not wish to include in a build.

File system folders

The VCS now allows you to check in external components without the file system folders being created within the repository; previously, any folders would have been

created. The old behavior can be restored by unchecking the Check-In preference "Ignore file system folders for external components".

Prompt for Options and Notes

The VCS will prompt you if you have not activated the Options and Notes tab when checking in/out. You can manage this new behavior on the VCS Options, Check Out & Check In tabs using the new "Prompt for Options and Notes" option.

External Components

oXML

Removing Invalid Characters

A static function $removeinvalidcharacters has been added to the oXML component to remove invalid characters from XML data.

- **$removeinvalidcharacters**
  $removeinvalidcharacters(&xData,iEncoding,iReplaceChar,&cErrorText) discards or replaces invalid XML characters in xData and returns the number of characters discarded or replaced, or NULL and cErrorText if an error occurs.
  xData: The data to check for invalid XML characters.
  iEncoding: The encoding of the xData parameter. One of kUniTypeAuto (defaults to kUniTypeUTF8 if encoding cannot be determined), kUniTypeUTF8, kUniTypeUTF16[BE|LE], kUniTypeUTF32[BE|LE], and kUniTypeNativeCharacters.
  iReplaceChar: Either -1 meaning discard invalid XML characters or the value of the character (0-255) used to replace the invalid XML characters.
  cErrorText: Receives error text if an error is returned.

Invalid XML characters are deemed to be characters less than space, that are not tab, carriage return or linefeed.

Omnis Datafiles

Exporting Double Quotes

Double quotes are now exported as a pair of double quotes when enclosing exported text in quotes (see RFC 4180 point 7). If double-quotes are used to enclose fields, then a double-quote appearing inside a field must be escaped by preceding it with another double quote. For example: "aaa","b""bb","ccc".

What's New in Omnis Studio 10.0

The Omnis Studio 10.0 release provides a significantly enhanced Method Editor allowing free-type entry of Omnis code, new features to make your apps accessible for people with disabilities, and a new conversion tool for migrating your Omnis datafile based apps to SQL. In addition, there is a new Remote Object class for executing methods on the client, and a new JavaScript editor to allow you to enter a whole block of JavaScript code directly into a method.

The following new features have been added to Omnis Studio 10.0:

- **Method Editor**
  The **Method Editor** in Omnis Studio 10.0 has been significantly enhanced, and now allows you to enter Omnis code directly into each command line in a method, with additional help from the Code Assistant; in addition, a new **JavaScript Editor** allows you to add or edit whole blocks of JavaScript code into client executed methods (the new embedded text editor also works for SQL & TEXT blocks)

- **Accessibility**
  a comprehensive set of features to support the **Web Content Accessibility Guidelines** (WCAG 2.0) to help to make your applications more accessible, primarily for people with disabilities; specifically, a number of **ARIA** properties have been added to most JavaScript controls which allow by screen readers to describe the controls, plus tabbing between and inside fields has been improved to allow end users to navigate a form entirely from the keyboard or by voice

- **Omnis Datafile Migration**
  there is a new tool to convert your applications that use the Omnis datafile to SQLite or PostgreSQL; following conversion the **Omnis DML** commands in your old library will be retained and will execute against the selected database, automatically and without modification to your database code; this provides you with a path towards conversion of your Omnis DML based applications to SQL, which will also allow you to convert parts of your app to the web or mobile devices

- **New & Enhanced JavaScript Controls**
  There is a new JS **Toolbar** control for remote forms, and a new External component **iCalendar** for managing calendar events in remote forms (also window classes); plus several of the other JavaScript components have been enhanced, including new shortcut keys for **Edit** controls, new properties for the **Segmented** and **Progress** controls to improve appearance, and the ability to upload multiple files in the **File** Control; plus several enhancements for **Data Grids** including the ability to validate data in cells, to copy selected data from the grid, and to fix a number of columns on the left of the grid

- **Remote Debugger**
  Remote Debugging allows you to debug and test your Omnis libraries and code located on a remote server

- **Remote Objects**
  Remote Object classes are Object classes that can be instantiated and executed

entirely on the client in a client-executed method in a remote form; this will allow you to make your web & mobile apps more agile and efficient

- **Web and Email Worker Objects**
  there are new OW3 Worker Objects for JavaScript (node.js), POP3 email, CRYPTO encryption or decryption, and HASH for hashing data, plus Secure FTP support has been added to the FTP worker object

Method Editor

The **Method Editor** has been significantly enhanced, including the *Code Editor* part (right-hand panel) which now allows you to enter Omnis code directly into each command line in a method, replacing the point-and-click style of code entry available in previous versions of Omnis Studio. When combined with the existing *Code Assistant* (introduced in Studio 8), and many new keyboard shortcuts, the enhanced Method Editor will allow you to write Omnis code quicker and more easily.

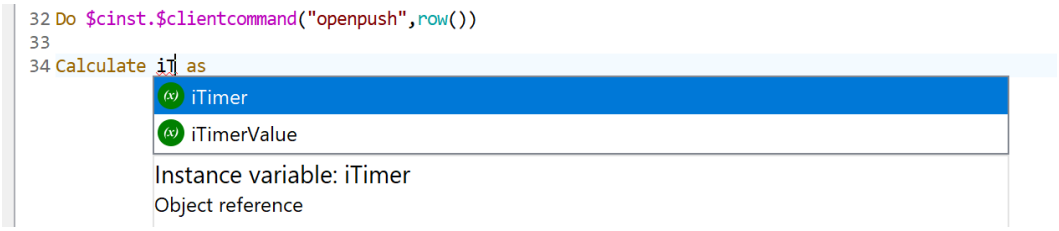
The old Command selection palette, which appeared at the bottom of the Method Editor window, and included the Command list and parameter options, has been removed from the Method Editor. To enter a command, you now tab or click into the first line of a method, then type the first few letters of a command name, and select it from the Code Assistant popup list using the arrow and Return keys. As you type a command or line of code, the Code Assistant will provide more help with command syntax, variable names, parameters and command options.

To enter a line of code in the new Method Editor:

- Click or tab into an *empty method line*; the insertion point should be at the start of the empty method line; you can press Ctrl-N to create a new line under the current line
- Type the first few letters of the command you want to enter; for most commands you will only need to type 2 or 3 characters (you can ignore case and leave out any spaces in the command name)
- As soon as you start to type, the Code Assistant will drop down automatically showing a list of commands that match the characters you have typed; you can press **Tab** to select the first/selected command in the list, or use the **Arrow** keys to navigate up or down the list, and press Return/Enter to select a command
- Having selected the command, you can start to fill out its parameters; again, you can type the first few characters of a variable name or parameter and select it from the Code Assistant help list

For example, to enter a calculation using the *Calculate* command, you can type "ca" (note lower case) and press the **Tab** key to select the *Calculate* command from the

help list, which should be the first command in the list. The insertion point should now be between ‘Calculate’ and ‘as’. Type the first few characters of the variable name or notation you want to enter, select the variable or notation from the help list (you can press Tab to select the first item in the list):



Once you have selected the variable name for your calculation, you can press Tab to go to the end of the command line, in this case, after the ‘as’, and then enter the calculation, including any functions or notation.

In all other respects the Method Editor behaves the same as in previous versions, including the Chroma coding which has been greatly enhanced with an updated theme. The following sections provide more detail about entering commands in the enhanced Method Editor.

Tokenization

Omnis is a tokenized language, and that remains the same in Studio 10.x. This means that all method text has a single canonical representation generated from the tokenized representation of the code. As you enter text into the new Code Editor, Omnis tokenizes the code and then updates the editor with the canonical representation. For example, this means that extra whitespace will be deleted, and attempts to indent the code using a non-default indent will have no effect. In addition, each command must occupy a single line, and command lines do not wrap. Each level of indent corresponds to two spaces.

Entering Code

To enable the free type entry of Omnis code in the Method Editor, there has been a number of enhancements in the editor interface, including enhancements in the **Code Assistant,** new Help panels at the foot of the editor window, new and updated menu options, and a whole raft of new keyboard shortcuts to speed up code entry.

Ctrl-space

The Code Assistant drops down automatically when you type a command name or some notation, but you can force the **Code Assistant** to open at other times. To open the Code Assistant manually, position the caret in the code text, press **Ctrl-Space,** and the text immediately before the caret is used to determine the contents of the Code Assistant help list.

One situation in which this is useful is if you cannot remember the syntax of a command: position the caret immediately after the command name, press Ctrl-Space and then down arrow, and you will see the command syntax in the Code Assistant help list.

Undo and Redo

The Method Editor now supports multiple levels of Undo, and Redo. (The multi-level Undo/Redo also applies to all Edit fields in the fat client and IDE.)

Commands

To enter an Omnis command the cursor must be on an empty line, and you can start to type the name of the command you need. As soon as you type the first letter, the **Code Assistant** will open automatically, displaying a list of commands starting with that letter: note that the command filters may limit which commands are shown, see below about the filters. As you type further letters of the command name, the Code Assistant

refines the list of available commands. In most cases you will only need to type the first 2 or 3 letters to locate a command. The text immediately before (to the left of) the caret is used to determine the content of the Code Assistant help list.

To select a command from the Code Assistant help list, you can press the **Tab** key to select *the first command displayed* in the list, or you can use the **arrow keys** to navigate up and down the help list and use **Return** to choose the selected command.

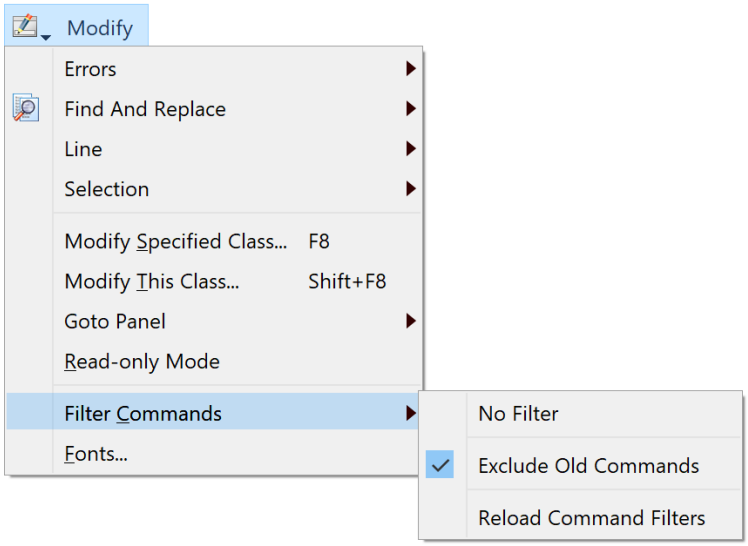
Assuming the cursor is at the end of the selected command name, you can start to enter its parameters, and the Code Assistant should pop up automatically at the insertion point whenever a variable name or parameter is needed.

Command Filters

The commands in Omnis perform many different functions, including many legacy features that are no longer required for creating web and mobile apps using the JavaScript Client. There is a new filter mechanism in the Method Editor to filter the list of commands that are displayed in the **Code Assistant** help list, primarily to remove any old commands, including those that allow you to manage Omnis datafiles.

Note you can still use the excluded commands in your code, and methods in converted libraries using these commands will continue to work – the filters just hide the commands from the Code Assistant help list.

The command filter is set under the **Filter Commands** submenu in the Modify menu: note this is only visible when the cursor is in the code entry area. The **Exclude Old Commands** filter is enabled by default, which excludes over 100 old commands, plus there are other filters available that exclude smaller subsets of commands. You can disable the current filter using the **No Filter** option, in which case all the commands available in Omnis will be shown in the Code Assistant help list.



The current filter option is saved with the Window Setup for the Method Editor: if the saved value is no longer present, the editor reverts to no filter and all commands will be shown in the Code Assistant.

The **Reload Command Filters** option reloads the filters from the commandfilters folder, without having to quit Omnis, which is useful if you have changed or added any filters.

Further Command Filtering

Normally, all commands matching the *first typed character* appear in the Code Assistant list, but you can limit or change which commands are shown depending on the number of characters typed – this may be useful if you want specific commands to always appear, instead of the default ones that appear first in the alphabetical list of commands. You can activate this further command filtering by enabling the **Use Minimum Lengths** option on the **Filter Commands** submenu.

The filtering enabled by the **Use Minimum Lengths** option is controlled in the file min_command_characters.json (located in the studio folder) which specifies the minimum number of characters to be typed for a specified command.

The JSON file contains an object, where each member name is either a command name, or a regular expression matching a set of command names. The value of each member is the minimum number of characters to type (default 1 if there is no match for a command). In the following example, Quit method appears as soon as you type Q, whereas the other Quit commands require you to type Qu, and the Queue commands require you to type Que:

```
    "^Queue.*": 3,
    "Quick check": 4,
    "Quit method": 1,
    "^Quit.*": 2
```

Regular expressions must start with ^, otherwise the entry is treated as a full command name.

If the file is not present in the studio folder, or if it cannot be loaded for some reason (e.g. invalid JSON syntax), the Use Minimum Lengths menu item is hidden.

Omnis loads the file min_command_characters.json at startup, and when you execute the **Reload Command Filters** command on the Filter Commands menu.

Editing the Command Filters

You can create your own filters, or change the ones provided, to change the commands that are shown in the Code Assistant help list. If you wish to adapt the default filter, you are advised to make a copy of it, rename the copy, then edit and save the new file.

The command filters are located in a folder called ‘commandfilters’ in the Studio folder: the default filter is called ‘Exclude_Old_Commands.json’. Each file in this folder is loaded in the Filter Commands submenu, and the name of the JSON file is used as the menu option name. (You can examine the contents of each filter file to see which commands they exclude from the Code Assistant help list.)

The content of each JSON file is an object with a single member named “exclude”, listing any commands that are to be excluded from the Code Assistant help list. The exclude member is an array, and each array entry is the exact command name (case insensitive).

You can exclude groups of commands using a regular expression to match command names: in this case, you need to anchor the regular expression to the start, using ^. For example, to exclude all old MSM… commands, you can create a filter file with the following contents (name the file ‘Exclude_MSM_Commands.json’):

```
{
    "exclude": [
        "^MSM.*"
    ]
}
```

As well as creating an exclude filter, you can create a filter to only *include* certain commands, although in practice this might only be useful if you want to use a very small subset of commands in the Method Editor (since all commands that *are not included* are excluded). To create an include file, create a new filter file containing an “include” object, and add any command names to be included, e.g. to only include Do and Calculate (and exclude all other commands!), the filter should contain:

```
{
    "include": [
        "do",
        "calculate"
    ]
}
```

The default or initial filter is set in the 'currentCommandFilter' option in the 'codeAssistant' section of the config.json file: if this is empty, or the command filter files or folder are removed, then "no filter" is selected.

You need to select the **Reload Command Filters** option in the Modify menu to load any new or edited filters into the Filter Commands submenu.

Case and Omitting Spaces

You can ignore the case of all command names, so you can always start to *type a command name in lower case.* Furthermore, if the command name includes spaces, *you can omit the space(s),* which will speed up command selection in the Method Editor.

Whether or not you include the space can, however, *determine which command is selected* by the Code Assistant: this is important for the *Do…* commands, for example. Typing do<space> will immediately enter a *Do* command (and the insertion marker will be ready to accept the calculation) and close the Code Assistant. Whereas, to select the *Do method* command, you can type dom<tab> (note no space), or to select the *Do async method* command, you can type doa<tab>. This is quicker than typing just 'do' and then selecting the command you want from the droplist in the Code Assistant.

Another example would be in the case of the If… commands. Typing if<tab> will immediately close the Code Assistant and enter an *If* calculation command, whereas, to select the *If canceled* command, you can type ifc<tab>. Similarly, typing on<space> will select the *On* event command, while typing ond<tab> will enter the *On default* command.

Tab key

You can use the Tab key to tab between the parameters of all of the commands in the method. This is an easy way to navigate through the commands, skipping command names and keywords and moving the insertion point to the next available position. You can also use the Tab key to select the first or selected line in the Code Assistant: in this case, if you select a method, such as Do List.$define, the opening and closing parenthesis () will be added automatically and the cursor is placed between the parenthesis.

Construct Commands

If you enter a construct command using the Code Assistant, such as *If,* it will add the end construct command automatically, in this case, *End If.* You can use Undo to remove the end construct command added automatically if it is not required.

The Method Editor checks for missing associated commands as you edit, e.g. *If* with no *End If*, or *For* with no *End For*.

Command Options

The command options that were entered using a *checkbox* or *radio button* in the command palette in previous versions can now be entered directly from the keyboard. To enter options in the new Method Editor, you either type (at the start of the command parameters if nothing is required prior to the options, or if you have entered a parameter, you type space and then (. This causes the Code Assistant to pop up the available options. After you select an option in the list, the Code Assistant adds it to the text and automatically closes the option list using). If there are no more options

available, it positions the caret after the). Otherwise the caret is before the) and you can type comma to add a new option: when you type comma, the Code Assistant pops up a list of remaining available options.

Class Names

To enter a quoted class name, you can press Ctrl-Space when the caret is positioned after a double quote (or some text following a double quote) and select the name from a quoted list of non-system class names in the current library.

Side by Side Editors

You can open two instances of the method editor to show *two methods from the same class,* for example. You can open a second copy of the method editor as follows:

- ❑ Press the **Shift** key while performing an action that opens the method editor, such as double-clicking on a remote task.
- ❑ Use the **Two Editors Side By Side** option on the method editor **View** menu.

Omnis opens a second editor, next to the current editor window, so that each editor uses half of the available screen space. On Windows, this means the available space in the main Omnis application window, and on macOS, it means the available space on the current monitor less the menu bar or toolbars.

When two editors are open, the same method in each class can be selected in both editors, but the editor in the background does not display the method: it displays the text "This method is being edited in another method editor".

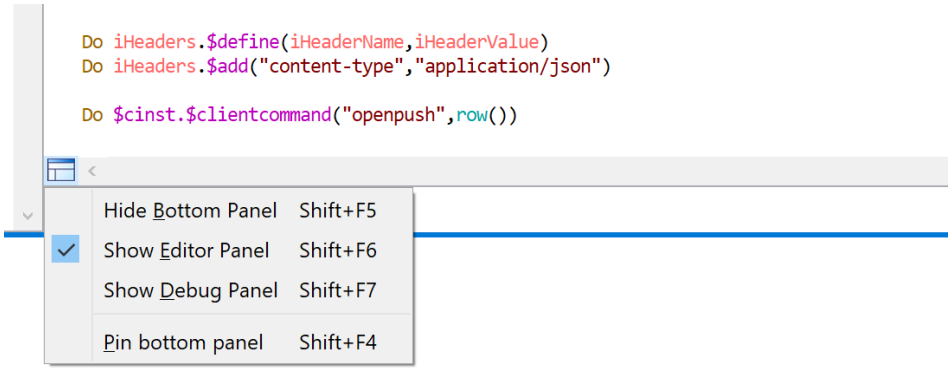
The editor in the background keeps up to date with changes in the foreground editor, e.g. when you add or delete a method, the method list in the other editor updates.

There is a keyboard shortcut for the Two Editors Side By Side command, which defaults to Alt+S on Windows and Cmd+Opt+S on macOS.

Panels

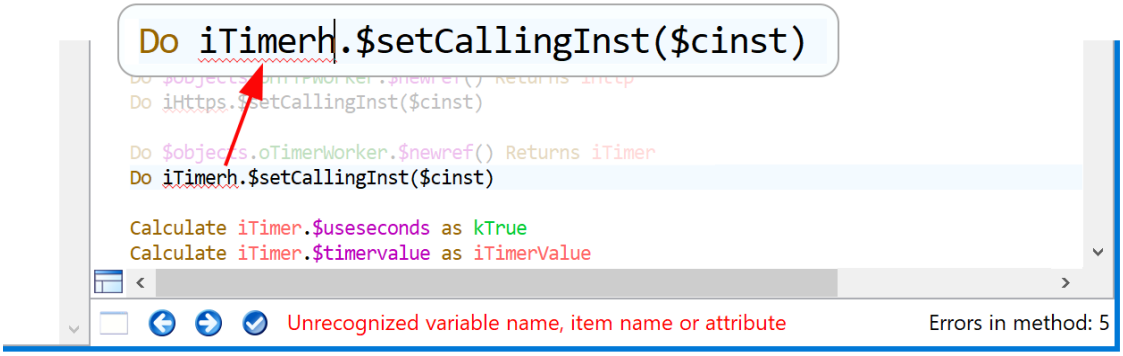
The Method Editor has two panels below the code entry window. The *debug panel* works as it did in previous versions. The *command entry panel* has been replaced with the editor panel.

You can select the panel (or hide it) using the small popup menu immediately to the left of the horizontal scrollbar at the bottom of the code entry window.



Editor Panel and Errors

As you enter code, Omnis tokenizes the entered code and provides real-time feedback that indicates if the method code is valid. Valid method code is syntax-colored, whereas invalid method code is partially syntax-colored, and the invalid component(s) in the method line underlined using a colored wavy line (the color is taken from the current theme or set in the “badsyntaxcolor” $appearance preference).



The editor panel at the foot of the Method Editor window displays the number of method errors, and when the caret is positioned within text causing an error, it displays the error text.

The editor panel has three buttons that allow you to handle errors. The Next and Previous error buttons (forward and back arrows) navigate through the errors in the method. The Fix error button (check mark) allows you to fix certain errors and will only be enabled when the caret is positioned in some text for an error. The Fix button is enabled to allow the following errors to be fixed:

- "Unrecognized variable name, item name or attribute" and "Unrecognized variable name": Pressing the Fix button opens the Create Variable dialog.
- ") missing": Pressing the button adds the)
- "Partly entered keyword": Pressing the button completes the keyword

In addition, the editor draws a red marker in the vertical scrollbar for each method line containing an error. The marker in the scrollbar is positioned so that when the method line containing the error is scrolled to be the first displayed line, the top of the scrollbar thumb lines up with the top of the marker. (Note that this is why the vertical scrollbar always allows scrolling even if all method lines fit within the editor window.)

Create Variable Prefixes

When you encounter the "Unrecognized variable name" error when entering code in the editor, you can press the Fix button to open the Create Variable dialog to create the variable. You can now specify the initial scope for a new variable using a predefined *prefix.* For example, you can begin the variable name with "i" to create an instance variable, or "p" to create a parameter. The default variable prefixes are:

Prefix	Variable scope
i	Instance
c	Class
p	Parameter
l	Local
t	Task

The prefixes allowed in the Create Variable dialog can be configured in the Omnis configuration file (config.json) using a new entry called "createVariableScopePrefixes" and located in the 'codeAssistant' section in config.json:

```
"createVariableScopePrefixes": [
    "i:Instance",
    "c:Class:",
    "p:Parameter",
    "l:Local",
    "t:Task"
],
```

The Create Variable dialog processes these entries in array order, and as soon as it finds a scope that is allowed for the method being edited (e.g. instance variables are only allowed for class types that have instances), where the first part of the entry value case insensitively matches the start of the variable name, it uses the configured scope (the second part of the entry value after the colon) to set the initial scope suggested by the dialog. If no prefix match occurs, the scope suggested is local.

Create Variable Suffixes

As well as setting the scope of a variable, using a prefix, you can specify the data type of a variable using one of a set of predefined *suffixes.* For example, you could enter the name “iData**Row**” which would create a variable of type Row, typing “iDataList” would create a list, and typing “iVarRef” would create an item reference. The default variable suffixes are:

Suffix	Variable type
Row	Row variable (kRow)
List	List variable (kList)
Ref	Item reference variable (kItemref)
Date	Date variable (kDate)
Obj	Object variable (kObject)
Bin	Binary variable (kBinary)

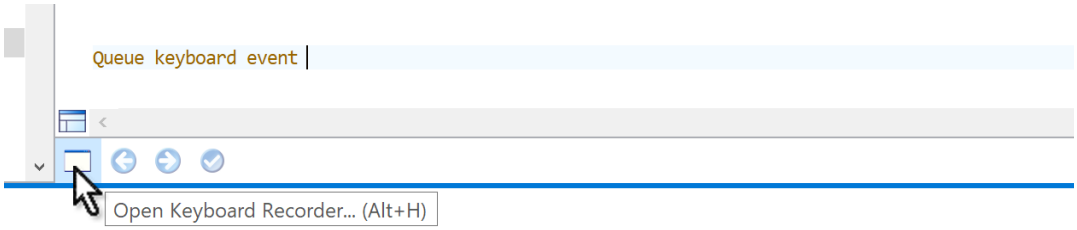
The suffixes allowed in the Create Variable dialog can be configured in the Omnis configuration file (config.json) using a new entry called “createVariableTypeSuffixes” and located in the ‘codeAssistant’ section in config.json:

```
"createVariableTypeSuffixes": [
    "Row:kRow",
    "List:kList",
    "Ref:kItemref",
    "Date:kDate",
    "Obj:kObject",
    "Bin:kBinary"
],
```

Omnis strips any consecutive digits from the end of the desired variable name, and then compares (case independently) the end of the resulting name string against the suffixes in the config.json array (strings before the colon in each array entry). If there is a match, and if the variable type is suitable (e.g. it is not a non-client executed type when creating a variable for a client-executed method), then the initial type is set using the type constant after the colon.

Editor Helper dialog

In addition to the error reporting, there is a button to open the Helper Dialog, which is context specific. This button is disabled when the context means there is no helper dialog. If a helper dialog is available, the button is enabled, and its tooltip changes appropriately: pressing Alt+H will open the Helper Dialog.

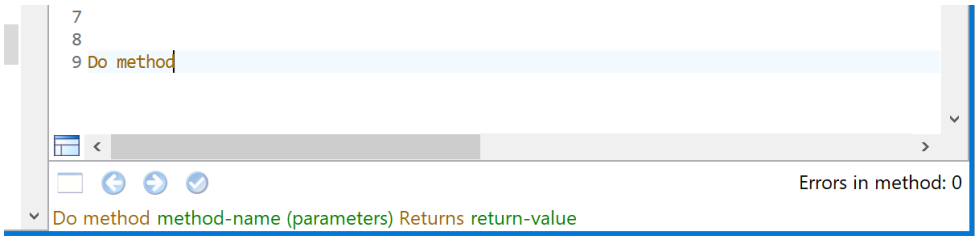


The editor Helper Dialog is enabled in the following cases:

- When the caret is positioned in the parameter field of the *Queue keyboard event* command. In this case, the helper dialog allows you to record keys.
- When the caret is positioned in the title parameter of the *Working message* command. The helper dialog is the working message configuration dialog.
- When the selection includes only *JavaScript:* commands (in a client executed method). The helper dialog button will open the JavaScript editor. All JavaScript: command lines in the same contiguous block are selected, and their JavaScript is then editable using the popup editor. When the popup editor closes, Omnis replaces the selected JavaScript: commands with JavaScript: commands containing the contents of the popup JavaScript editor.
- When the selection includes only *Sta:* commands (for entering a SQL statement on multiple lines). The helper dialog opens the same external editor for JavaScript but in SQL mode allowing you to enter a SQL statement over multiple lines.

Command Syntax Help

You can view the full syntax for a command, including all its parameters and options, in the Help panel at the bottom of the editor window. This type of help is displayed once you have selected a command from the Code Assistant list, or you have typed the command name in full – as you reach the last character of the command the syntax help is shown. For example, if you type the *Do method* command, its syntax is show in the Help panel at the bottom of the editor window.

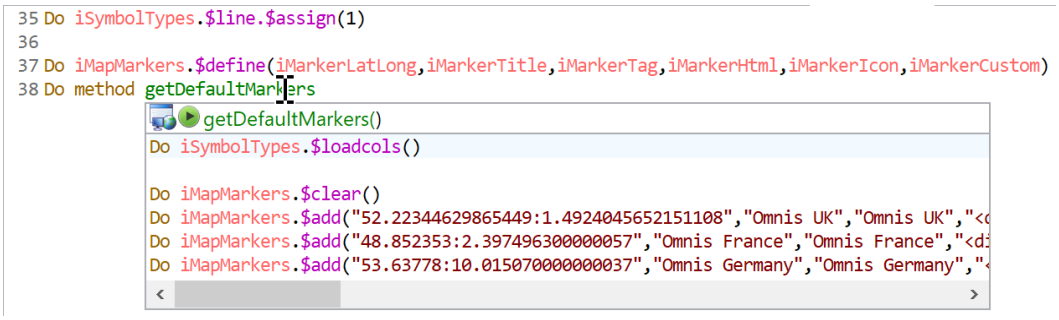


You can hide the command syntax by unchecking the "Show Syntax Strings" option on the View menu.

Method Tooltips

Tooltips are displayed when you hover the pointer or I-beam over a method name in the Method Editor, including methods listed in the Method tree list on the left of the editor window:

or method names that are being called in your code (assuming Omnis can identify the method being called).



The method name and its code are displayed in the tooltip and you can scroll longer methods using the mouse or trackpad. You can hold down the Shift key to keep the tooltip window open when you move the pointer, which allows you to scroll the window more easily.

The Method tooltips provide a useful preview of a method, without having to switch away from the current/selected method you're working on. You can dismiss the method tooltip by moving the mouse away from the method name and tooltip, or by pressing Escape.

There are three entries in config.json that control the size of the Method tooltips:

- "maxWidthOfMethodTooltip": 500 (value in pixels)
- "maxHeightOfMethodTooltipGeneralInformation": 100 (value in pixels)
- "maxVisibleMethodLinesInMethodTooltip": 20 (number of lines)

When used with the method tree list, the maximum width used is the width of the code edit text field if it is wider than the maxWidthOfMethodTooltip config item.

Maximum Number of Methods

The maximum number of methods allowed per class has been increased from 501 to 4096.

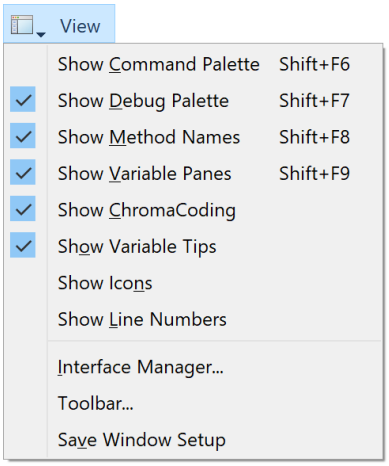
Menus and Keyboard Shortcuts

The Method Editor menus have been re-worked and improved for Studio 10 and include several new commands or options. The keyboard shortcut keys for some options have changed and these are listed below where they occur – there is a summary of the keyboard shortcuts at the end of this section. Where there are significant changes, an image of the menu from Studio 8 and Studio 10 is shown, so you can compare them.

View Menu

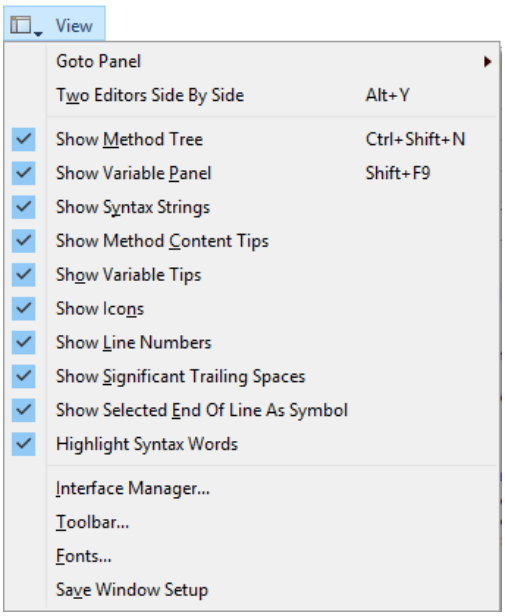
The View menu in the Method Editor has several changes or additions; some of the new options are discussed elsewhere. The **Show Debug Palette** and **Show Chroma Coding** options have been removed; the latter option has been replaced by a more comprehensive set of color options stored in the default theme in the IDE (you can change the theme in the Studio Browser Hub under **Options,** including a dark theme which may be more suited to working in the code editor).

Studio 8



Show Method Names Shift+F8

Studio 10

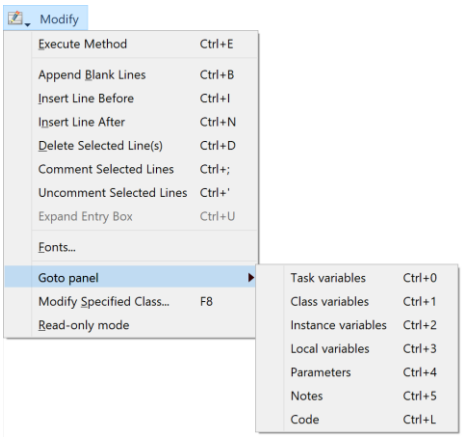


Show Method Tree Ctrl/Cmnd+Shift+N

Goto Panel

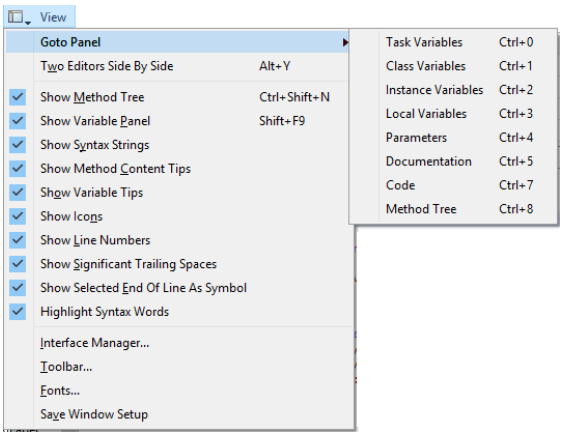
The **Goto Panel** option lets you select a different pane in the Variables list (with keyboard shortcuts Ctrl+0 to 5). It also lets you switch the insertion point to the **Code** text entry area (Ctrl+7) ready to enter some code, or back to the Method Treet list from the code entry area (Ctrl+8).

Studio 8



Goto Code Ctrl+L

Studio 10



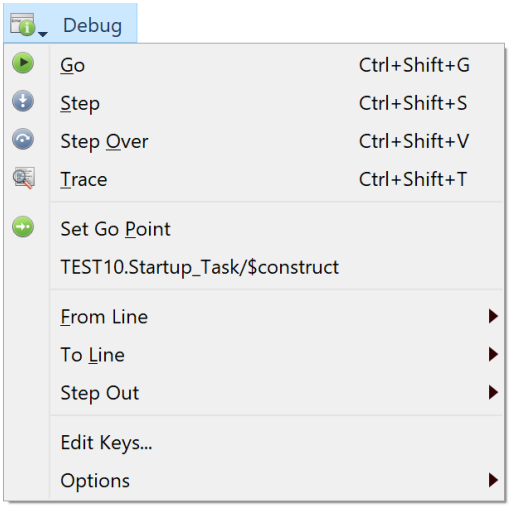
Ctrl+7

Note the Goto options were on the Modify menu but are now on the View menu

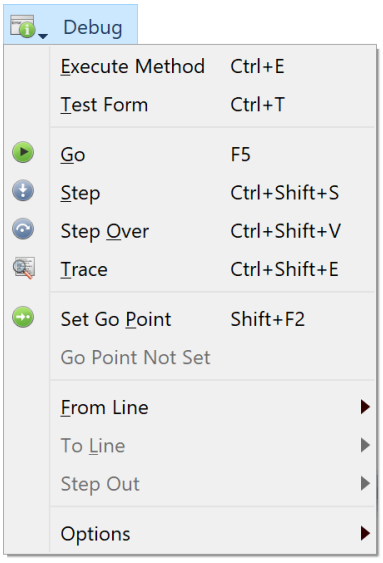
Debug Menu

The **Execute Method** and **Test Form** commands have been added to the Debug menu. Note that the shortcut keys for Go, Trace and Step Out>>Go have changed, plus you can **Set Go Point** using Shift+F2. The From Line, To Line and Step Out options, and the debug **Options,** are unchanged.

Studio 8



Studio 10



Go Ctrl+Shift+G

F5

Trace Ctrl+Shift+T

Ctrl+Shift+E

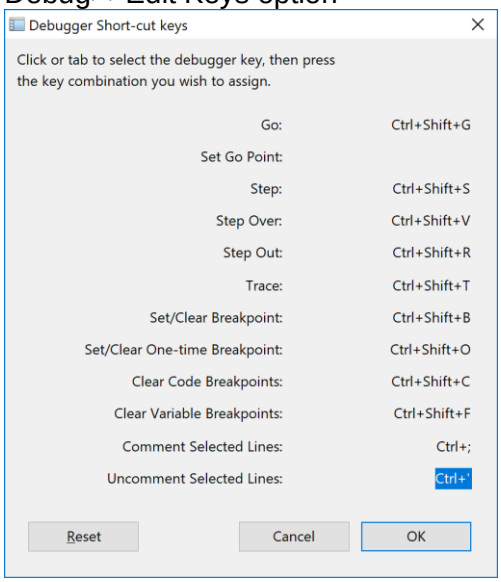
Having entered code using the new text entry method, running the debugger on your your code is exactly the same as in pevious versions of Omnis. You can set the Go point, then click Go or Step in to execute your code: as your code executes the

debugger will scroll automatically to the center of the code entry area when the current line is positioned at around 75% of the visible lines.

The **Edit Keys** option has been removed and replaced with a section in the $keys property in the Omnis preferences, which you can edit in the Property Manager.

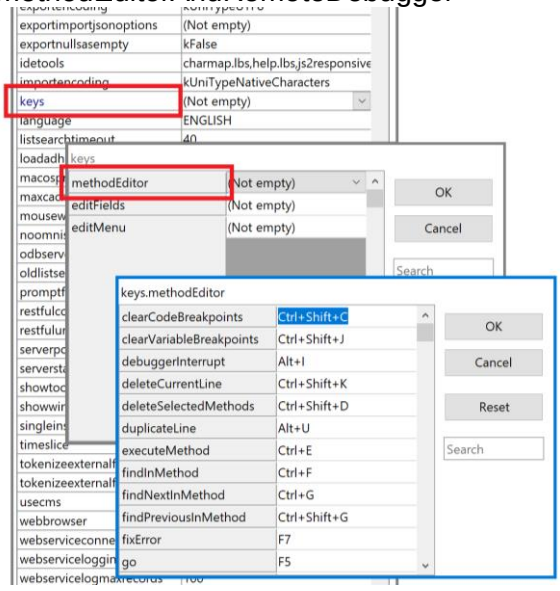
Studio 8

Debug>>Edit Keys option



Studio 10

$keys Omnis preference > methodEditorAndRemoteDebugger



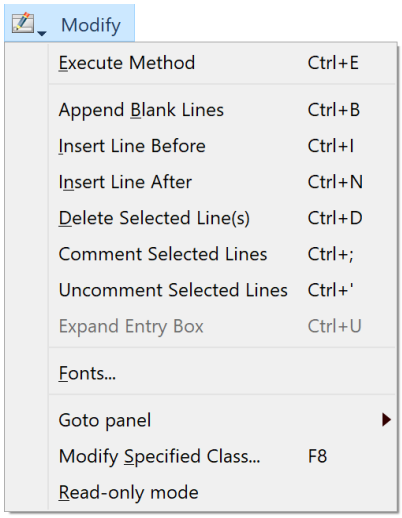
See later in this section for information about changing the keyboard shortcuts.

Modify Menu

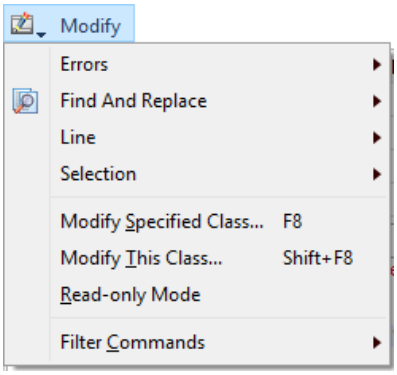
The **Modify** menu contains new submenus for **Errors** and **Find And Replace.** The **Execute Method** option has been moved to the Debug menu, while the Goto panel and Fonts options have been moved to the View menu. The various Line options have been moved to the **Line** submenu.

The Comment & Uncomment options have been merged and moved to the **Selection** submenu. For classes which have an associated editor, the **Modify This Class** option opens the class editor, such as a JavaScript remote form; the shortcut key is Shift+F8.

Studio 8

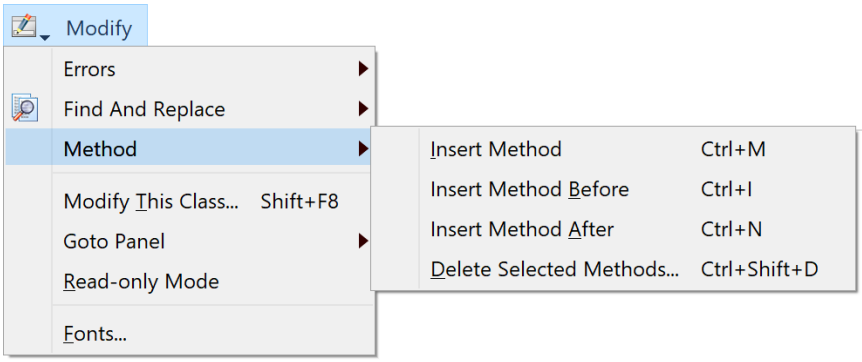


Studio 10



There are additional entries that depend on the focus, as follows.

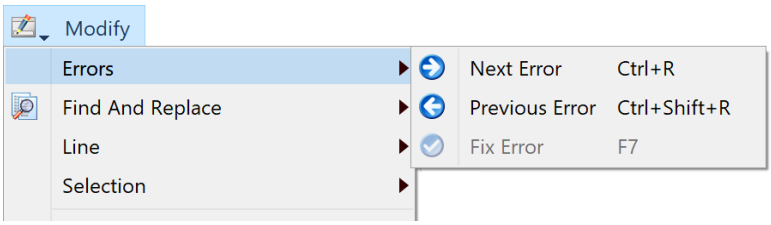
- ❑ If the focus is on the Method Tree (on the left, containing a list of methods for the class), the Modify menu contains a submenu called **Method,** which allows you to **Insert Method** (at the end of the method list, or *Before* or *After* the current method), or **Delete Selected Methods**



- ❑ If the focus is on the Code text entry area (on the right), the Modify menu contains submenus called **Line** and **Selection:** see later in this section for info.

Errors Menu

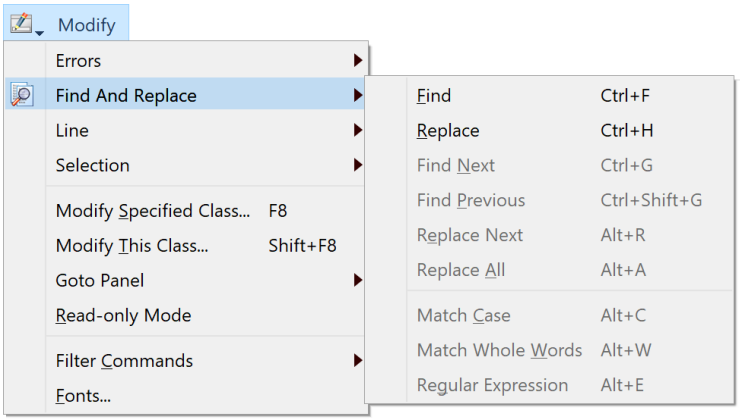
The Modify>>**Errors** submenu is new and contains **Next error, Previous error** and **Fix error** commands, that can be used instead of the buttons on the editor panel. These also have keyboard shortcuts.



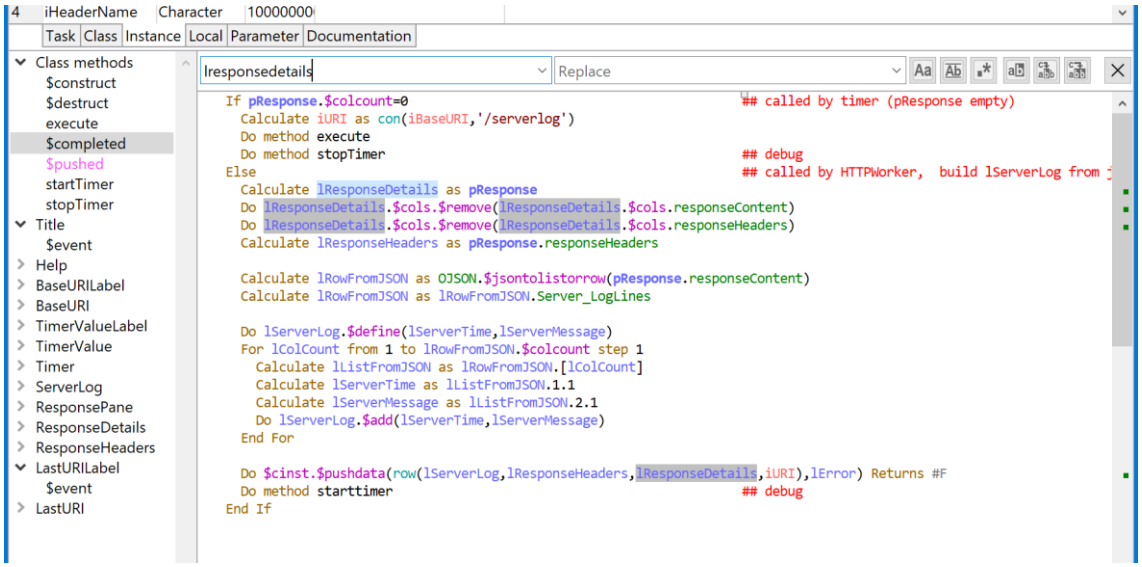
Note that when the focus is on the method tree, this menu is only present when only one method is selected.

Find And Replace Menu

The Modify>>**Find and Replace** submenu is new and allows you to perform a local find and replace on the *method text for the current selected method*. Note that when the focus is on the method tree, this menu is only present when only one method is selected. This menu also allows you to toggle options such as match case.



The menu commands also have keyboard shortcuts, such as Ctrl+F to open the Find panel, Ctrl+H to open Find and Replace, or Ctrl+G to find next. When you first select the Find or Replace command, the editor opens a panel immediately above the code entry field, where you can enter the find (and replace) text.



The panel also contains buttons that perform the same operations as the menu items, as follows:

Aa	Match case	Alt+C
Ab	Match whole words	Alt+W
.*	Use regular expressions	Alt+E
ab	Find Next or Previous	Ctrl+G or Ctrl+Shift+G (to Find Previous, you can shift click the button)
	Replace next	Alt+R
	Replace all	Alt+A

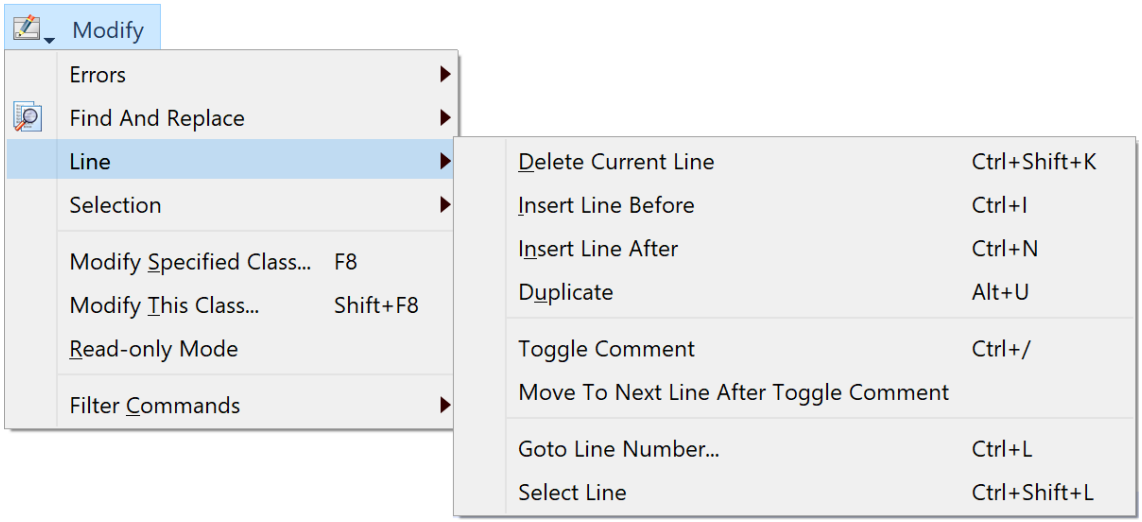
As you type characters into the find text field, the code text area dynamically updates to reflect the found text. It highlights the found text (e.g. the text 'lresponsedetails' is searched and highlighted in the above image), and it also adds a green marker to the vertical scrollbar, in a similar way to the error marker, drawn to the right of the error marker.

After closing the Find (or Find and Replace) panel, you can still use Find Next and Find Previous, although the editor no longer highlights all matches.

Line Menu

The options in the **Line** submenu replace several options in the Modify menu in previous versions, including Insert Line After, Insert Line Before, and Toggle Comment. Note that you can Right-click on the current or selected lines of code to open a context menu with similar options.

The Comment and Uncomment line options available in previous versions have been merged into a single **Toggle Comment** command, which has the single keyboard shortcut Ctrl+/ for commenting or uncommenting lines.



The Line Menu contains the new option **Select Line** which selects all the text in the current line (triple-clicking on a line also selects the line), and the **Delete Current Line** option which deletes the current line (containing the cursor or word selection), or all lines where multiple lines are selected.

The **Duplicate** option duplicates the current line (if no text is selected) or all selected lines, and places the duplicate line(s) immediately below the original line(s). The command also selects the duplicate text, which then allows you to use repeated Duplicate commands to generate multiple copies.

The **Goto Line Number** option opens a box to allow you to enter a line number to go to. You can show line numbers in the code area using the **Show Line Numbers** option in the View menu.

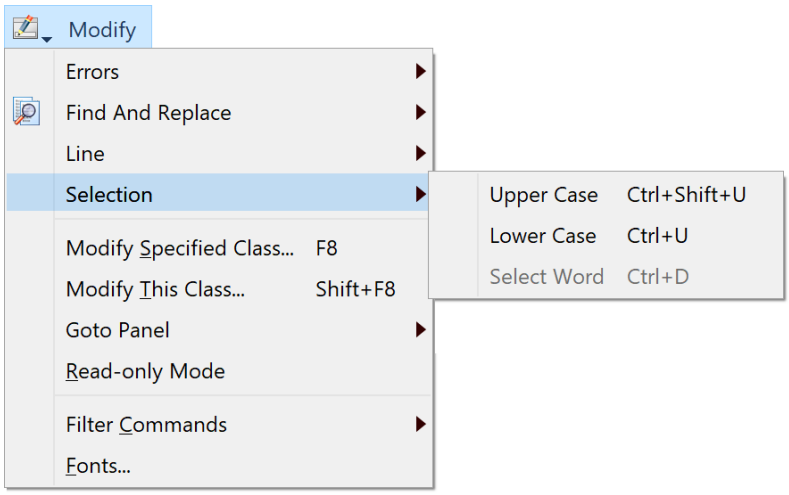
Commenting / Uncommenting Lines

You can comment or uncomment a single method line by clicking anywhere in the line (or you can select the whole line) and selecting the **Toggle Comment** option, or press the Ctrl+/ shortcut. To comment or uncomment multiple lines, you need to select all the lines and then use the Toggle Comment option: in this case, all the affected lines will remain selected after toggling their comment state. Commenting a *single empty line* does not select the commented line: in this case (and when "Move to next line after toggle comment" is off, see below), the caret is positioned after the comment character and the space, ready for you to type the comment.

You can force the cursor to move down to the line after the commented/uncommented line or block of selected lines by enabling the **Move To Next Line After Toggle Comment** option in the Line menu (the option is off by default): the state of this option is saved with the Window Setup.

Selection Menu

The Modify>>**Selection** submenu contains new commands **Upper Case** and **Lower Case:** note that these options only change case for text that does not have a single canonical form, e.g. text in strings.



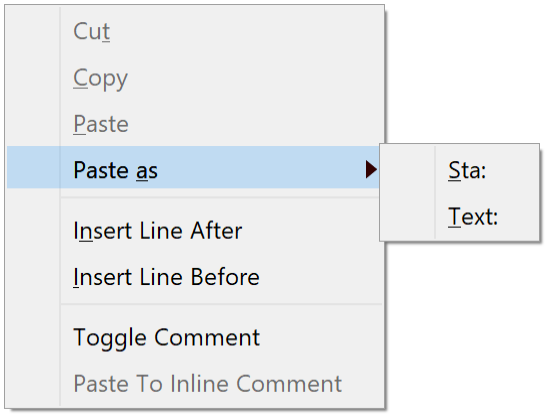
In addition, the Selection submenu contains the option **Select Word** which selects the word containing the insertion point, or where the insertion point is at the beginning or end of a word; in the latter case the word to the right or left of the insertion point is selected.

Word Selection

You can double-click on a word to select it, or double-click and drag the pointer to select multiple words. If you double-click on a single word that is enclosed in quotes (e.g. like the foo in *Calculate lcVar as "foo"),* the quotes *will not be* selected. In previous versions the quotes would have been selected, but if want to enable the old behavior you can set a new option "entryFieldsIncludeQuotesWhenSelectingWords" in the "defaults" section of config.json to true; the option defaults to false which enables the new behaviour.

Method Editor Context Menu

The Method Editor context menu (opened when you right-click on the Code text area) has a new hierarchical menu called **Paste as.** You can use this to paste multiple lines of text from the clipboard into Sta:, Text: or JavaScript: commands. The Paste as hierarchical menu items are enabled when the caret is positioned on an empty line.



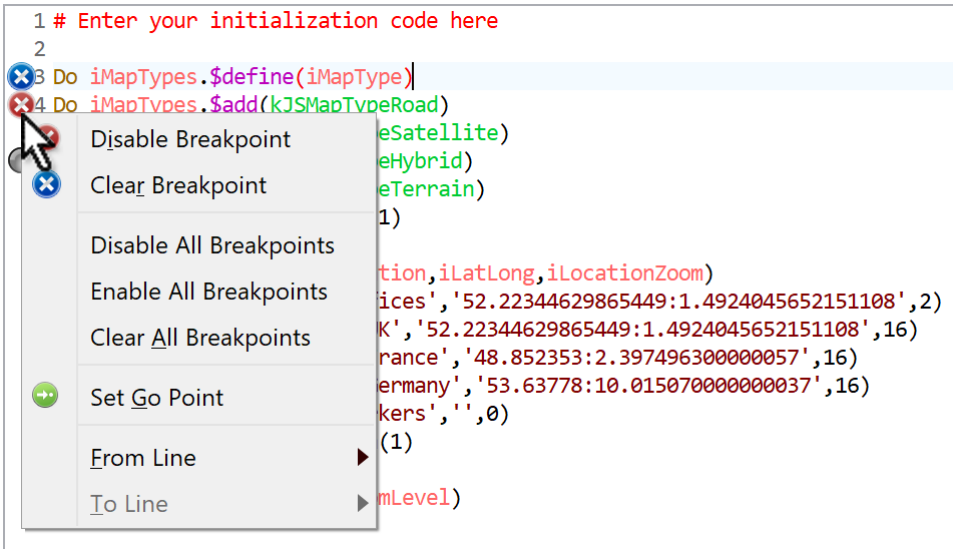
Setting Breakpoints & the Breakpoint Context Menu

You can set a Breakpoint, a One-time breakpoint or the Go Point using the pointer (to click on the code margin) and the keyboard:

- ❑ You can set a **Breakpoint** using a single click in the left margin of the code editor, next to any line of code where you want the breakpoint.
- ❑ You can set a **One-time breakpoint** using Ctrl/Cmnd+click next to the line of code
- ❑ You can set the **Go point** using Shift+click next to the line of code

(Existing users should note that you can no longer set the Go point by double-clicking in the left-hand margin.)

Alternatively, you can use the **Breakpoint** context menu by right-clicking in the left margin of the code editor, next to any line of code, and selecting the option.



In addition, the Breakpoint context menu shows Delete and Disable/Enable breakpoint commands when there is a breakpoint already set for the line. It also shows the commands to Clear/Disable/Enable all breakpoints. And if there is an active stack, as well as set Go point, there is a command to Clear the stack.

Keyboard Shortcuts

A major enhancement in the Method Editor has been to add many new keyboard shortcuts to allow you write Omnis code from the keyboard alone, without having to use the pointer. With this in mind, the most significant menu options in the Method Editor now have keyboard shortcuts, including most of the options in the **Modify** and **Debug** menus, as well as the **Find and Replace** options.

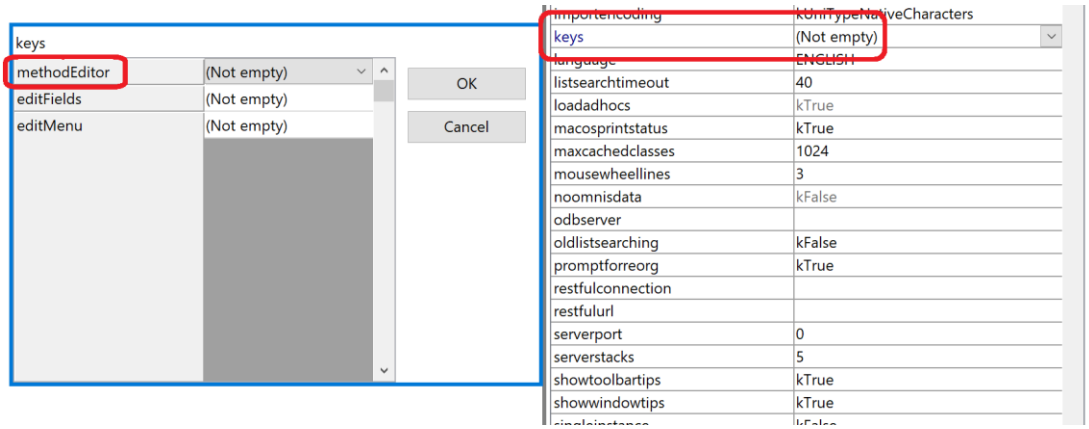
A complete and updated list of keyboard shortcuts has been added to the Studio 10.1 section earlier in this manual.

Keyboard Shortcut Configuration

The keyboard shortcuts are stored in a new property in the Omnis Preferences ($prefs) called **$keys,** which you can edit in the Property Manager to change the keyboard shortcuts. Note this feature replaces the Edit Keys option on the Debug menu in previous versions, and it also contains the keyboard shortcuts for Edit fields and the Edit menu.

The first time you edit $keys *and press OK*, Omnis generates a file called **keys.json** in the Studio folder, that records the configuration of the keyboard shortcuts (as listed above); if you don't make any changes in $keys the default keyboard shortcuts will be stored in keys.json.

You can edit the Shortcut Keys options by selecting **$keys** in the Property Manager (find it under the Omnis Preferences in the Studio Browser), then select **'methodEditorAndRemoteDebugger'.**



To edit a value, you can use the **Delete** or **Backspace** key to clear the current shortcut, and then type the desired shortcut key combination. You can use all the standard Key modifiers (Ctrl, Cmnd, Alt, Option, Shift, etc) as well as all the letter and number keys, plus the numbered Function keys. In addition, you can use the **Enter** and **Return** keys in conjunction with Ctrl/Cmnd, and optionally Shift or Alt/Option, for method editor menu shortcuts.

The $keys preference also contains the shortcut keys for Edit fields (editFields), which are documented under the JavaScript Edit Control, and the Edit menu (editMenu) which has the following shortcut keys:

Shortcut Key	Description	keys.json item
Ctrl+Y	Redo last operation	Redo
Ctrl+Shift+F	Find and Replace	findAndReplace
Ctrl+Shift+G	Find Next	findNext

Language Syntax

There are a number of changes to the Omnis language syntax that facilitate direct text entry of commands, and which enable the new Omnis Studio 10.0 language parser to function properly.

Language Keywords

The following language keywords can no longer be used as variable names:

as	at	flag
for	from	into
on	returns	sec
step	to	until

During library conversion, any variable names using these keywords are appended with an integer starting at 1.

Options

Omnis now stores the order in which "checkbox" and "radio button" command options are specified as part of the method command (remember that the Omnis language is tokenized, and does not store raw text as entered by the developer). This allows you to enter the options as text in any order.

The "Select matches (OR)" and "Deselect non-matches (AND)" options of the Search list command have been renamed to "Select matches OR" and "Deselect non-matches AND". This prevents the parentheses in these option names from interfering with language parsing.

Braces

Braces have been removed from all commands, except for commands like OK message, which require three components (a field name or square bracket calculation, options and a calculation). For these commands, when they use square bracket calculations, you must escape ( ) { } characters in the calculation *outside square brackets* if there is no text *after* the parentheses. In this case, these characters need to be escaped using square bracket notation, e.g. [‘(‘] escapes (.

Comments

The way comments are displayed has changed. The character used to show a comment is now # (hash) rather than ; (semicolon), and the inline comment marker is now ## rather than ;; (library conversion will change ; to #).

Changing the comment prefix character to # allows the language parser to work correctly when ; is being used as the function parameter separator (as there would otherwise be parsing confusion caused by a call with an omitted parameter).

Entering a new comment

To enter a new comment, on an empty line, you can type # and then the comment text, with or without a space after the #. You can also type ; to create a new comment, but the comment is marked with #.

To enter an *inline comment,* press the space key followed by ## at the end of a code line, and then enter the comment text. Inline comments are positioned over on the right of the code entry area: they are left-tab aligned according to a tab which is indicated by a small marker at the top of the code entry area: you can drag this marker to reset the tab position.

The Sta:, Text: and JavaScript: commands (that generate a text block) no longer allow inline comments (see note in *Library Conversion* section about inline comments for the Sta: command). This allows all text after a colon to be treated as significant text, and to be added to the text block, with the exception of the options string specifying the line delimiter for the Text: command.

Commenting and Uncommenting code

You can “comment” or “uncomment” the current method line (containing the cursor) or any selected method line or lines using the **Toggle Comment** option in the Modify menu, or using the keyboard shortcut **Ctrl-/** (forward slash) – note the same menu option or keypress can be used to both *comment* or *uncomment* method lines or comments as appropriate. Commented lines must have valid syntax to be uncommented, otherwise they will remain commented out.

Errors

As the new Method Editor allows any text to be entered, it is possible to enter and store commands that contain errors. Internally, these are stored in the method with a new command type, and will cause an error to be reported if you try to export the method to JSON, or if you try to execute them.

The Find and replace dialog has a new option (Only search method lines containing an error), which you can use to find commands with an error. When you check this option, the dialog also checks the regular expression option, and sets the find string to the regular expression “.*”.

In general, there should not be much need to leave erroneous commands stored in a method for very long - the editor gives immediate feedback about errors, so in practice it makes sense to fix them as you code. The Find and replace dialog option provides a means to double check that all is well with a library. Omnis Studio 10.0 takes this approach (rather than for example marking all classes with an error count) since errors should be very much an exceptional case once coding of a method is complete.

Modified Commands

The step interval for the *For* command is assumed to be 1, so when entering a *For* loop and you want the step interval to be 1, you no longer need to enter this. If you need a step interval other than 1 you need to enter this into your code.

Obsolete Commands

Some of the commands that were marked with 'OBSOLETE COMMAND' in previous versions (listed in the 'Obsolete Commands…' group) have been removed from the Omnis language and will be commented out in your Omnis code by the converter. The *Translate input/output* command has also been made obsolete and will be commented out.

The *Call method* OBSOLETE COMMAND will be replaced by the *Do code method* command and the method name.

There is a list of obsolete commands that have been deleted in this version in Appendix A in this manual.

Library Conversion

The changes in language syntax mean that Omnis performs a class-by-class conversion of a library created using Omnis Studio 8.1.x or earlier. The following items are converted:

- Some obsolete commands will be commented out. In previous versions these commands were marked with the "OBSOLETE COMMAND" suffix and listed in the 'Obsolete commands' group, and have now been removed from the Code Editor (but most will continue to work in the Runtime version of Omnis). Any commands that have been removed in this release and are commented out are listed in an appendix.
- The prefix for comments is now #, converted from ;
  A space is inserted after the # at the start of comments, therefore comments are # abc rather than #abc after conversion.
- Inline comments for JavaScript:, Text:, and Sta: commands Inline comments are no longer allowed, since *all the text after the :* is treated as part of the statement or text. Therefore, on conversion, all inline comments are moved to the next line and inserted as a standard comment (see below).
- Square bracket calculations (ctySquare, ctyParmlist4 etc) are converted so that any text outside square brackets does not contain unescaped characters () {}.
- Any instances of " ##" are detected in method lines and reported as a warning (they are probably not editable as the parser will treat the text after this sequence as the inline comment).
- Any variables which are named using a *language keyword* (see earlier for a list) are renamed, by appending an integer to them (starting with 1 until a new unique name in its context is created).

Inline Comments for JavaScript:, Text: and Sta: commands

By default, the conversion process will move all inline comments from JavaScript:, Text: and Sta: commands to the *next line in the method*, after the original line containing the inline comment. There are three new options in the "ide" section of **config.json** to allow you to control how inline comments are treated.

- **"libConverterAppendsDiscardedInlineComments"**
  When true (the default), if the inline comment would otherwise be discarded, the converter appends a comment command after the JavaScript:, Text: or Sta: command, containing the inline comment.
- **"libConverterAddsInlineCommentToStaCommandParameter"**
  Note that if you use "libConverterAddsInlineCommentToStaCommandParameter"

to convert inline comments for Sta: commands, then this option will not affect Sta: commands; see below.

- **“libConverterInsertsDiscardedInlineComments”**
  moves and inserts the inline comment *before the original line* containing the inline comment (however, if libConverterAppendsDiscardedInlineComments is set to true, libConverterInsertsDiscardedInlineComments is ignored).

Inline Comments Sta: commands

If you want to keep inline comments as part of the SQL text for Sta: commands, you can set the item “libConverterAddsInlineCommentToStaCommandParameter” in the ‘ide’ section of config.json to a formatting string, e.g. " -- %" or " /* % */". Omnis replaces the first % place-holder in the formatting string with the inline comment, and appends the resulting string to the parameter of the Sta: command. Note that if the resulting text does not tokenize, e.g. if the inline comment contains text like [#S333] which does not tokenize, then the comment will be discarded.

If you leave “libConverterAddsInlineCommentToStaCommandParameter” empty (or supply a string that does not include the % character), then Omnis will discard the inline comment when converting Sta: commands.

SQL comments for the Sta: command are colored, including /* */ and – comments. The “syntaxColorProbableSQLComments” option in the ‘ide’ section of config.json is enabled by default, but can be set to kFalse to disable coloring.

Conversion Logs

The converter adds an entry to the Find and Replace log that allows you to quickly navigate to each change made by the converter by double-clicking on a line in the log. In addition, the converter writes a log file to the ‘conversion’ folder in the logs folder in the data part of the Studio tree. The log file provides a more permanent record of the changes applied to the converted library. Note that Omnis does not write log entries to record where spaces were inserted at the start of comments.

JSON generated libraries

When Studio 10 imports JSON generated with Studio 8.1, it parses methods using the old Studio 8.1 parser, and then applies the same conversion steps as above to the imported classes. Changes applied by this conversion are written to the Find and Replace log only.

Syntax Coloring

The chroma coding in the Method Editor has been extended, with several new colors and styles, which can be changed by changing the theme in the Hub>>Options in the Studio Browser, or configured by editing the $appearance preference in the Property Manager (these are stored in appearance.json and the various theme files): the new colors are in the IDEmethodSyntax group in the appearance.json file. The following new theme colors and styles have been added:

Color option	Description
optioncolor optionstyle	command options (corresponding to check boxes or radio buttons in the old editor, e.g. Sound bell for OK message)
constantcolor constantstyle	Constants (e.g. kTrue)
eventparametercolor eventparameterstyle	event parameter variables
functioncolor functionstyle	built-in and external functions

Color option	Description
hashvariablecolor hashvariablestyle	hash variables
localvariablecolor localvariablestyle	local variables
parametervariablecolor parametervariablestyle	method parameter variables
instancevariablecolor instancevariablestyle	instance variables
classvariablecolor classvairablestyle	class variables
taskvariablecolor taskvariablestyle	task variables
notationcolor notationstyle	built-in notation attributes
badsyntaxcolor	bad method syntax indicators
methodothertextcolor	The color for all other text with no specific syntax color, e.g. separators, dots, etc.
methodhighlightcolor	The color of selected method text in the Method Editor when the control displaying the method text has the focus
syntaxwordhighlightcolor	Color used to highlight syntax elements, e.g. click on a variable name in the code editor to highlight all mentions of the variable
methodcurrentlinebackgroundcolor	The background color used to display the line containing the caret in the Method Editor
methodhighlightnofocuscolor	The color of selected method text in the Method Editor when the control displaying the method text does not have the focus
methodeditorcodebackgroundcolor	The background color for the method editor code area

Note that the existing theme colors variablecolor and variablestyle now only apply to file class variables (field names) and other components of a variable string, e.g. a list column name.

Syntax Highlighting

When you click in a syntax element (variable, notation name, command name (not block commands) or function name), the code editor performs a find and highlights instances of the element in the current method (note the find highlighting will override the syntax highlighting if the Find or Find and Replace panel is displayed).

```
Do iList.$define(#S1,#S2)
# Set current list iList
# Build externals list
For lNum from 1 to 10000
  If mod(lNum,100)=2
    Do iList.$add(con("Line",iList.$linecount),iLongChar)
  Else
    Do iList.$add(con("Line",iList.$linecount),#CT)
  End If
End For

Calculate iRow as iList.1
```

The view menu contains the option “Highlight Syntax Words” which is checked by default. There is a new color option"syntaxwordhighlightcolor” in the “IDEmethodEditor” group in the $appearance Omnis preference, and stored in the appearance.json file.

Printing Methods

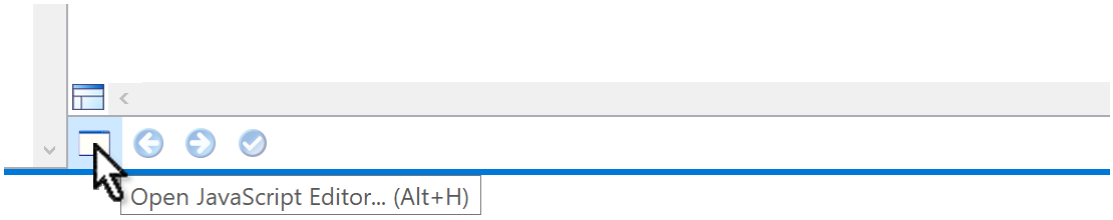
When you print methods using the File menu, Omnis now uses the syntax colors from the default theme (which is designed for a white background). You can turn off this behavior (and print everything using black text) by setting the entry “printMethodsWithSyntaxColors" in the “methodEditorAndRemoteDebugger" section of config.json to false.

JavaScript: Editor

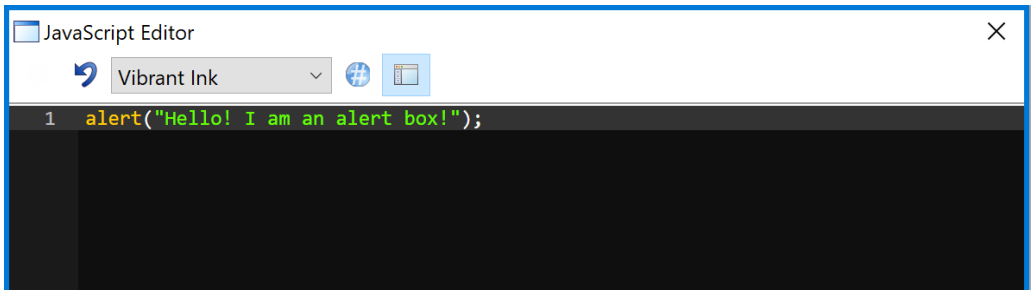
In addition to the main interface changes in the Method Editor, a JavaScript editor has been added to the Method Editor to allow you to enter a whole block of JavaScript code directly into the *JavaScript:* command, rather than line by line as in previous versions. The new JavaScript editor will popup whenever you edit a command line containing the *JavaScript:* command. The editor also allows you to enter a SQL statement if the *Sta:* command is selected; in this case, the editor will switch to SQL mode.

To edit or enter some JavaScript, click into or tab to a *JavaScript:* command line in the text entry panel, or select a whole *JavaScript:* command line or multiple lines, and either

- Press Alt+H to open the JavaScript editor, which in this case is the same as clicking on the Helper dialog button at the bottom of the Method Editor window
- Or select *Open JavaScript Editor* from the **Modify>>Selection** submenu



The content of the JavaScript editor is formed by concatenating the contiguous JavaScript: command(s) that are selected in the list. This allows you to edit or insert a contiguous sequence of these commands as a block. Omnis selects unselected lines in this contiguous block when it opens the window, so all the lines are selected when viewed behind the editor window. When you have finished editing the JavaScript: text, you can close the editor window and Omnis replaces the selected JavaScript: commands with the new content, creating a JavaScript: command line for each line of JavaScript.



The editor window allows you to change the theme of the displayed text, and to revert to the original text.

Spaces & End of Line Characters

There are two new options in the Method Editor **View** menu to allow you to show Trailing Spaces and End Of Line characters when editing text in the new popup JavaScript or SQL text editor:

- **Show Significant Trailing Spaces**
  If true, the editor displays trailing spaces for the JavaScript:, Sta: and Text: commands as the Unicode sp symbol.
- **Show Selected End of Line As Symbol**
  If true, the editor displays the end of line character as Unicode symbol cr when the end of line character is selected. This allows you to see if an end of line character will be added to the clipboard by a cut or copy, for example.

Both of the new options default to true and are saved with the window setup.

Trace Log

The *Send to trace log* command now includes the name of the method that issued the command in column 1. Double clicking on the trace log line takes you to the code line that issued the command.

In addition, the *Send to trace log* command now has the "Always log" option. If specified, the command will always log the message even if $nodebug is true for the library or the local debugger is disabled (this option is ignored for a diagnostic message).

Error Processing

There is a new library preference, $clib.$prefs.$errorprocessing, that controls how Omnis behaves when it would (in Studio 8.1 and earlier) either enter the debugger (if available) or report the error with an OK message.

- **$errorprocessing**
  A kEP... constant that indicates how unhandled errors in methods belonging to this library are processed. Values of the kEP… constant are:
  kEPreport: Report the error by opening the debugger if available or by displaying a message box
  kEPlogStackAndReport: Log the call stack to the trace log and then report the error by opening the debugger if available or by displaying a message box
  kEPlogStackAndContinue: Log the call stack to the trace log and then continue execution with the next method command

The default value of $errorprocessing after converting a library to Studio 10.0 is kEPreport.

The call stack written to the trace log is drawn using the bad syntax color from the appearance settings. The first line contains the error code and error text, and then subsequent lines in turn show the call that invoked the previous line. You can double-click on a line to open the method at the relevant method line, provided that the library is not marked as always private and the class is not protected. The call stack excludes

entries from methods running in tasks marked as IDE tasks which have their code in an always private library.

Dynamic Methods & Objects

The handling of dynamically added or modified methods, and dynamically added and removed objects has been improved.

- The stack list has a new menu item, to detach the debugger from an instance. Previously this was only possible by force-closing the current debug instance.
- The debugger tree lazily updates to show new or deleted objects in the current debug instance: typically, this means it updates either when the debugger window comes to the front, or while you are stepping through code.
- When an instance closes, or an object is removed, Omnis deletes any breakpoints set in a method in a freed temporary instance field method, and removes all of its methods (if any) from the method editor history stack used by the back and forward navigation buttons.
- Omnis marks each temporary method (i.e. instance method), using a new icon so you can recognise these easily in the tree. If you edit such a method, the edits are saved with the instance, and will be lost when the instance destructs.
- You cannot rename a temporary instance object shown in the method editor tree.

Accessibility

This release contains several enhancements to support the Web Content Accessibility Guidelines (WCAG 2.0) which will help to make your applications more accessible, primarily for people with disabilities. These guidelines have been adopted by many government agencies and guarantee an acceptable level of access to information and services via websites and applications for people with disabilities. You can read the following pages to gain a basic understanding of the WCAG requirements:

https://www.w3.org/WAI/standards-guidelines/

The WCAG implementation in Omnis Studio calls on the ARIA specification, which according to W3.org is "**Accessible Rich Internet Applications** (ARIA) defines a way to make Web content and Web applications more accessible to people with disabilities. It especially helps with dynamic content and advanced user interface controls developed with [various web technologies]," which includes technologies such as the JavaScript Client in Omnis Studio.

In practice, this means we have added various ARIA compliant properties to the controls for JavaScript remote forms which you can use in your web and mobile apps to support end users with disabilities. These properties will be read automatically when the screen reader capabilities are enabled in the end user's browser or mobile device. (For testing, we have used ChromeVox by Google, but there are many other screen readers for Chrome and other browsers.)

Accessibility Properties

Most JavaScript controls have a set of basic ARIA and other accessibility properties which are interpreted by the screen reader in the browser. The ARIA properties in Omnis map closely to their equivalent ARIA attributes in HTML.

General properties

Several of the JavaScript controls have the following ARIA properties, while some other controls have additional properties (listed below). These properties are designed to work in a similar way as their equivalent ARIA attributes in HTML.

- **$arialabel**
  the text for the aria label, which is used when a text label is not visible on the form. If there is a label for the control, use the $arialabelledby property instead
- **$arialabelledby**
  the name of a control to act as a label for this control; for example, you could enter the name a label object to link it to the control. A value in $arialabelledby will override the value in $arialabel
- **$ariadescribedby**
  the name of a control used to describe this control: similar to $arialabelledby, but could be used to provide more information or a longer description about the control

You should note that JavaScript controls now have an $active property which works alongside $enabled allowing you to make controls active, inactive, enabled, or disabled, which helps you control accessibility and tab order in your remote forms. See the JavaScript Components section for information about $active.

Image based controls

You can assign an Alt text value to image-based controls, such as Picture and Activity, using the $alttext property:

- **$alttext**
  a short text to describe the appearance or function of an image, and equivalent to the "alt" attribute in HTML; this property is relevant for controls that contain an image or have a significant visual appearance, such as the Picture and Activity controls.

Page panes and Landmarks

So-called "Landmark Roles" in standard accessibility guidelines allow you to identify different areas of a form to allow screen readers to describe the structure of the page to end users. You can define Landmarks in your Omnis JavaScript remote forms using *Page panes* and by assigning the appropriate value to a new $landmark property for each pane: the options for the new property correspond to the same keywords used for landmarks in the accessibility guidelines (Main, Navigation, Banner).

- **$landmark**
  specifies a role to make the page pane an ARIA landmark region, a kLandmark… constant with kLandmarkNone as the default.

The Landmark options are:

Landmark option	Description
kLandmarkMain	A "Main" landmark which identifies the primary content of the remote form
kLandmarkNavigation	A "Navigation" landmark which identifies an area containing navigation type control or list of links used for navigation
kLandmarkBanner	A "Banner" landmark which identifies an area usually at the top of the form, possibly containing logo, company or application name and search box

kLandmarkContentinfo	A “Contentinfo” landmark which typically identifies common information at the bottom of a form
kLandmarkComplementary	A “Complementary” landmark which many contain supplementary information or further links, such as a sidebar
kLandmarkForm	A “Form” landmark which identifies an area containing a number of input controls or other form controls
kLandmarkSearch	A “Search” landmark which typically would contain a Search field and button
kLandmarkNone	No landmark definition

Label controls

You can link a Label control to a specific Edit control, or you can tag a label as one of the HTML header types, using the following properties:

- **$labelfor**
  links a label to a control. If you use this with some controls such as the Edit control, the linked control will get the focus if the label is clicked. It can be used in addition to $arialabelledby.
- **$tagtype**
  can be used to set a label’s HTML tag type to one of the header types (<h1> etc.) which would allow the end user to navigate to different sections of a form: the default value is kJSLabelTypeLabel, which is a standard untagged label, and the other values include H1 to H6 for the header types.

Control text

If a control has some text assigned (e.g. a button), the screen reader will read out the text by default, therefore it is not always necessary to assign the ARIA properties to describe such controls. For example, the text for a Button control will be read by the screen reader, if no ARIA properties are specified, however the value in $text will be overridden if you specify $arialabel or $arialabelledby.

Content tips

The Edit control has the $::contenttip property which is a text string which is displayed in the edit field when it is empty and before the end user has entered any text. This can be used in addition to the ARIA label properties, to help label the edit controls on your forms: note it is good practice to add labels to all the edit controls on your form to help with accessibility, so do not rely solely on content tips to describe edit controls.

Keyboard Accessibility

As well as the ARIA properties, the behavior when using various keys to navigate a remote form, or inside more complex controls, has been improved. For example, when the end user presses the Tab key, the focus will jump from one control to another in a remote (web) form, or for complex items such as a Tab bar, the Tab key will put the focus inside the control and the arrow keys can be used to move from one element to another. In addition, the Arrow keys can be used to interact with controls, such as dropdown menus, while Enter and Spacebar can be used to select options or items. The Page Up/Down keys can be used to scroll a form or long list which has the focus.

Tabbing Order

The $order property determines the tabbing order for the controls within a remote form; note this is not a new property but has an impact on accessibility. The value of $order for each control is assigned automatically as you add controls to the form in design mode, starting at 1 and increasing by 1 for each control (note the $order values do not change if you rearrange the controls on the form). You can change the $order value of

a control to change its tab order: when you change the value of one control, the value of other controls on the form will shuffle automatically.

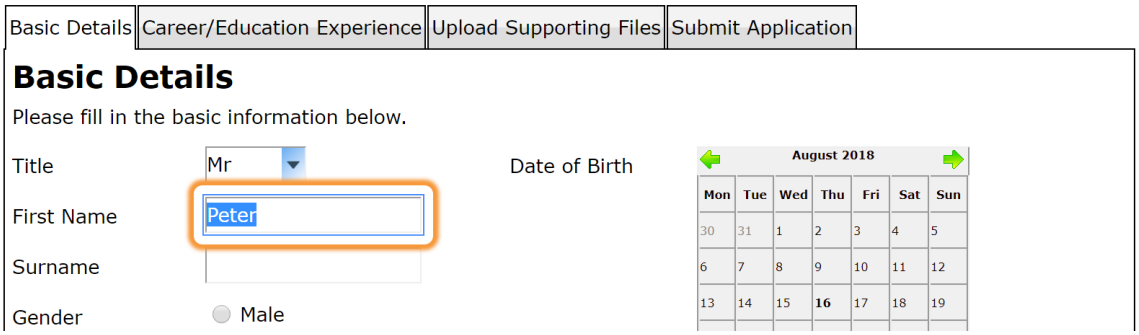
For increased accessibility in your applications, you should carefully consider the tab order of the controls in your forms. In general, it is good practice to make the tab order run consecutively, that is, from one control to the next in a logical order: this could be from left to right starting at the top of the form, but the exact order may depend on the specific functions of your app. The tabbing order of the controls in the form is also used by the screen reader to "read out" or describe the contents of the form, so it's important how you specify the tabbing order of the fields in your form. Once you tab into a container such as a page pane, the tab order takes you through all of the fields in the container, before tabbing out of the container.

The $startfield property specifies which field in a remote form will get the focus when the form is opened, overriding the control with its $order property set to 1; $startfield takes the field number as specified in the $order property of the control. Note this is not a new property but has an impact on accessibility, insofar as $startfield may not be the first field on the form, thereby going against most accessibility practice.

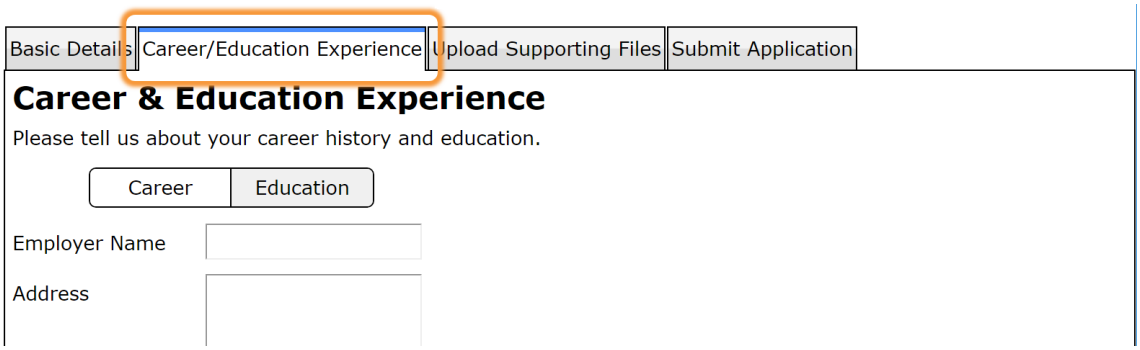
Form Example

With the ARIA labels specified and the correct tabbing order defined, the end user can navigate the controls on a form from the keyboard, and, in addition, the screen reader can describe each control or area of the form page in turn.

Consider the following JavaScript remote form. In the first image, as the end user tabs to the **First Name** edit field, the field border will highlight, the screen reader will say aloud: "First name, Edit text", and if there is a value in the field, as in this case, it will read that as well: "First name, Peter, Edit text".



Using the Tab key, the end user can move from one control or area of the form to another. Successive tab presses will enter the **Tab bar** at the top of the form, then the Right and Left Arrow keys can be used to move along the Tab bar, and the Return key can be used to select a tab. Once the tab is selected, the screen reader will describe the item selected: "Careers / Education Experience, Tab selected, 2 of 4".



Omnis Datafile Migration

There is a new Omnis Datafile conversion tool that lets you migrate the data in your Omnis datafiles to SQLite or PostgreSQL, ensuring the future stability and longevity of your OmnisSQL applications, and addressing several long-term issues with the old-style datafile architecture. For example, SQL statements are no longer limited to 64KB; compound Indexes may now contain columns of different types; and various SQL parsing issues including issues with GROUP BY and ORDER BY should also be resolved.

Following migration to a SQLite or PostgreSQL database using the new tool, the so-called “Omnis DML commands”* in your old library will be retained in the converted library and will execute against the selected database, seamlessly and automatically, rather than the old Omnis datafile. For example, commands such as *Prepare for edit* and *Update files* are kept and execute against the new SQLite or PostgreSQL database. This will allow you to switch from an Omnis datafile to a SQL database reasonably quickly and easily, without having to rewrite a lot of data handling code, which will make data storage more robust while giving you a route to convert your application to all SQL code.

*The commands that operate against Omnis datafiles and file classes have in the past been collectively referred to as the Omnis *Data Manipulation Language* or Omnis DML.

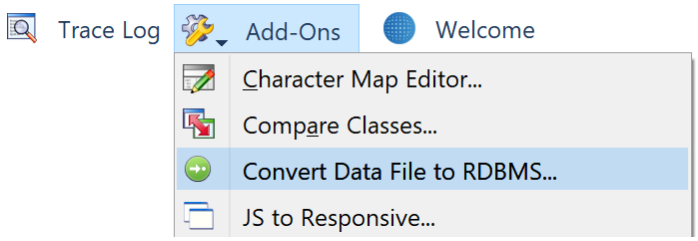
We would like to thank Nick Renders and Thad Bogert for their help in developing the DML emulation library.

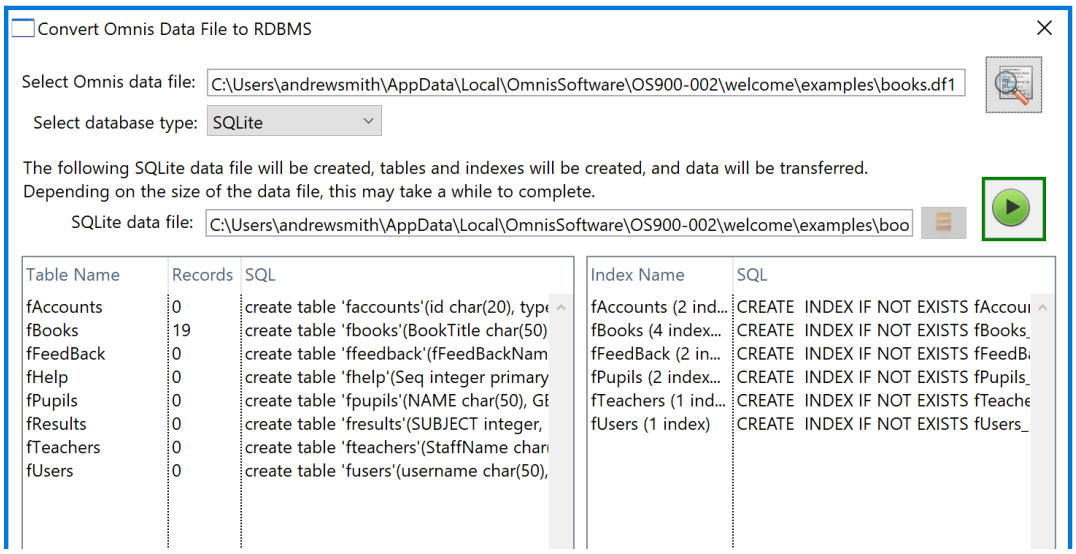
IMPORTANT: Backup Your Datafiles

IN ALL CASES, YOU SHOULD MAKE A SECURE BACKUP OF ALL OMNIS DATAFILES BEFORE OPENING/CONVERTING THEM WITH THE DATA FILE CONVERSION TOOL.

Converting Omnis Datafiles to SQLite

The new conversion option, called **Convert Datafile to RDBMS,** is available under the Tools>>Add-Ons option on the main Studio menubar or Toolbar, allows you to convert an existing single- or multi-segment Omnis datafile into a SQLite database file.





The conversion process copies all tables and indexes, and will also preserve any legacy file connections, replicating them using foreign key fields.

The conversion tool also has an option to convert an Omnis datafile to a PostgreSQL database (see DML emulation mode below).

New OmnisSQL DAM

The OmnisSQL DAM has been enhanced and now contains an internal SQLite object, giving it the ability to connect to old-style Omnis datafiles *and* to SQLite datafiles. Once converted to SQLite, simply modify your library code to connect to the SQLite datafile (.db file) in place of the old-style Omnis datafile (.df1 file), e.g.

```
Do omsqlSess.$logon('/Users/myUser/mydatafile.db','','') Returns #F
```

Note that no other code changes are necessary. When the DAM encounters the '.db' file-extension, it automatically connects to the SQLite datafile (and you may remove the old Omnis datafile).

The new DAM is designed to behave identically to the old-style DAM, i.e. it supports the same properties and methods. The SQL syntax and functionality supported by the new DAM is also exactly the same, i.e. there is no support for encryption, procedures, triggers or extended ISO SQL supported by SQLite. This is done to ensure backward compatibility with Omnis SQL.

Should you wish to adopt enhanced SQLite features, you will need to modify the library further so that it uses the SQLite DAM in place of the OmnisSQL DAM.

For details of the OmnisSQL Language Definition, please refer to the OmnisSQL V2 DAM API document. Please note that the new OmnisSQL DAM supports legacy datafiles and SQLite only. To support PostgreSQL, your application will need to be modified to use the PostgreSQL DAM in place of the OmnisSQL DAM.

Omnis DML Emulation

The Conversion library (omsqlconv.lbs) provides two functions. As well as converting old-style Omnis datafiles to SQLite or to PostgreSQL, the conversion library also works in the background emulating the Omnis DML commands, translating and executing them against the required RDMS. In addition, the conversion library emulates semaphores and reversible blocks where applicable.

To enable DML command emulation, it is necessary to set the *$mapdmltodam* Omnis library preference, for example, for SQLite:

```
Do $clib.$prefs.$mapdmltodam.$assign('SQLITEDAM')
```

or for PostgreSQL:

```
Do $clib.$prefs.$mapdmltodam.$assign('PGSQLDAM')
```

or you can set it in the Property Manager under the library properties.

Once assigned, the following DML commands will be executed either partly or entirely inside the conversion library, and against the specified SQL database:

- **Datafiles:**
  Close datafile, Close lookup file, Create datafile, Floating default datafile, Open datafile, Open lookup file, Prompt for datafile, Set current datafile, Set default datafile, lookup().
- **Data management:**
  Build indexes, Delete data, Drop indexes, Open runtime datafile browser, Rename data.
- **Changing data:**
  Cancel prepare for update, Delete, Delete with confirmation, Do not flush data, Do not wait for semaphores, Flush data, Flush data now, Prepare for edit, Prepare for insert, Prepare for insert with current values, Test for only one user, Update files, Update files if flag set, Wait for semaphores.
- **Files:**
  Clear all files, Clear main & connected, Clear main file, Clear range of fields, Clear selected files, Set main file.
- **Finding data:**
  Clear find table, Disable relational finds, Enable relational finds, Find, Find first, Find last, Load connected records, Next, Previous, Prompted find, Single file find, Test for a current record, Test for a unique index value.
- **Searches:**
  Clear search class, Reinitialize search class, Set search as calculation, Set search name, Test data with search class.
- **Sort fields:**
  Clear sort fields, Set sort field.
- **Lists:**
  Build list from file.
- **Others:**
  Begin reversible block, End reversible block, Quit all methods, $root.$getodbfilelist().
  Plus various sys() calls including sys(11), sys(139), sys(3000) & sys(3001).

Aside from setting the library preference, it should not be necessary to make any changes to the code in your library since the emulated DML commands will execute against the selected database automatically. When $mapdmltodam is set, the *Open datafile* and *Prompt for datafile* commands will also invoke conversion to SQLite if a datafile with the ".db" extension is not found. The *Create datafile* command will create a new SQLite datafile.

Logging on to a PostgreSQL Database

Whereas connection to a SQLite datafile simply involves changing the filename extension (from ".df1" to ".db", it is necessary to specify a logon configuration file when connecting to PostgreSQL. The logon configuration text file (".dfp") may contain any relevant PGSQLDAM session property, for example:

```
hostname=192.168.0.10
port=5432
username=postgres
password=postgres
database=postgres
```

Connection to PostgreSQL is made by specifying the logon configuration file, e.g.

```
Open datafile { C:\Users\MyUser\Desktop\pgconfig.dfp,internalName }
```

The *Prompt for datafile* command allows you to browse for a logon configuration file. The *Create datafile* command reads the logon configuration file and attempts to create the specified database.

DML Command Logging

Before Conversion:

Setting the library preference $mapdmltodam to 'LoggingOnly' invokes so-called 'dry-run logging' whereby DML commands continue to execute against the Omnis native data file. In this mode however, the omsqlconv library is also called and DML commands are logged to a file name commencing "dmlLog_" inside the Omnis writeable files folder. The resulting log file will indicate occurrances and locations of any DML commands that are emulated by the omsqlconv library.

After Conversion:

Should you wish to audit the use of DML commands after conversion, it is possible to log execution of these commands to a special table; *_dmllog*. This is useful if you are intending to migrate away from DML commands in the future. You can enable logging by calling sys(3000), and disable it by calling sys(3001). Note that calling sys(3000) clears any previous log results.

SQLite Data Bridge

The new **SQLite Data Bridge** is analogous to the old-style Omnis Data Bridge in that it provides Omnis clients with TCP/IP networked access to SQLite datafiles. The Write-ahead logging (WAL) model used by the data bridge also means that multiple concurrent connections to datafiles are supported.

SQLite data bridge connections may be shared by DML, OmnisSQL DAM, and SQLite DAM connections. The SQLite data bridge provides built-in encryption for data that is sent across the network. This is not to be confused with *datafile encryption,* a feature supported by the SQLite DAM only. Where the SQLite DAM connects to an encrypted datafile, this cannot be shared by OmnisSQL or DML connections.

Connection to the SQLite data bridge is made using a modified hostname of the form:

```
Do omsqlSess.$logon('sdb://192.168.0.10:5743/SQLiteTest','','') Returns #F
```

or via DML, for example:

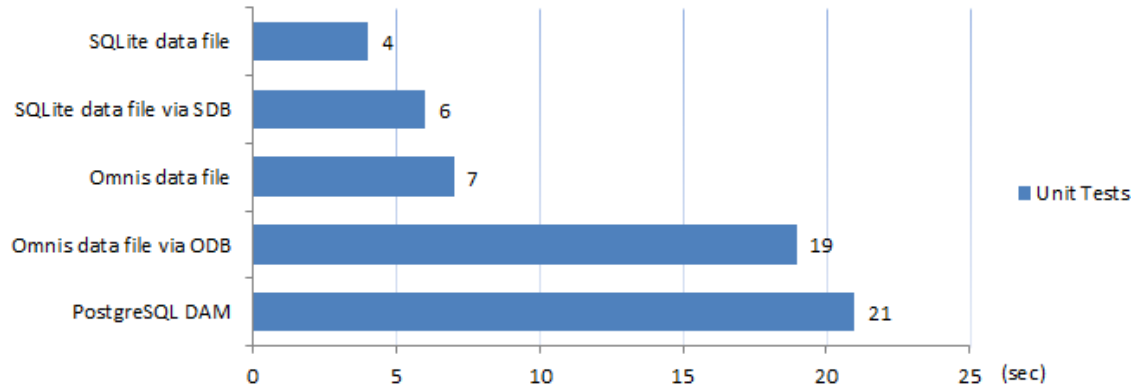
```
Open datafile { sdb://192.168.0.10:5743/SQLiteTest }
```

The conversion library also emulates the $odbfilelist command so that you can obtain a list of data source names, e.g.

```
Do $root.$getodbfilelist(myList, 'sdb://192.168.0.10:5743') Returns #F
```

Assessing Performance

Below are the results of unit tests made using the new OmnisSQL DAM to connect to SQLite and Omnis datafiles, both directly and using the corresponding data bridge. The same tests run using the PostgreSQL DAM are also shown. The unit tests insert ~ 2.5MB and receive ~20MB from a sample datafile. Test completion times are shown in seconds.



Results shown are for illustration purposes only.

JavaScript Remote Forms

Client Preferences

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

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

For example:

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

Do lPrefRow.$assigncols('omnis_pref1','iPref1',"session")
Do $cinst.$clientcommand("loadpreference",lPrefRow)
```

Sending Data to the Form construct

A new mechanism has been added to the Omnis JavaScript object to allow you to send data or content to the $constuct method of a remote form. The “omnisobject” in a remote form can now have two special attributes:

- **data-localstorage**
  A comma-separated list of preference names saved to localStorage (e.g. using the 'savepreference' $clientcommand), whose values should be sent to the $construct row in the form. They can be named "localpref_<prefName>"
- **data-window**
  A comma-separated list of members of the JavaScript 'window' object, whose values to send to the $construct row in the form. You can use dot notation to

access nested children. The columns returned to Omnis will be named "window_<memberName_childName_...>". Column names have a max length of 255 characters

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

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

Class Cache Logging

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

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

To enable the cache logger, the omnisobject can now have two optional attributes:

- **"data-logcaching"**
  If present, data will be collected on the caching of class data, etc in localStorage. This can be accessed by querying the JavaScript object jOmnis.omnisInsts[0].cacheLogger. It has methods getCacheLog() and printLocalStorage() to provide useful information in the browser console. If given the value "verbose", it will print caching messages to the console as they occur.
- **"data-onlycacheclasses"**
  If present, cache only the class data for the specified classes in localStorage. A comma-separated list of Remote Form classes whose data should be cached. In the format "<library name>.<form name>". E.g: "myLib.jsForm1,myLib.jsForm2" #STYLES is handled separately, per-library. To enable caching of styles, add an entry "<library name>.#STYLES"

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

Form Layout Events

The event handling when the layout changes in a JavaScript remote form has changed, with the addition of a new orientation parameter in evScreenOrientationChange, and a change in behaviour for evLayoutChanged.

All form layout types now trigger **evScreenOrientationChange** when their layout changes; this applies to kLayoutTypeResponsive, kLayoutTypeScreen, and kLayoutTypeSingle type forms. The event parameter pOrientation has been added to the evScreenOrientationChange event which will have a value of kOrientPortrait or kOrientLandscape depending on the *resulting* orientation of the form.

Only remote forms of type kLayoutTypeScreen trigger the evLayoutChanged event when their layout changes.

Layout Breakpoints

New remote forms now have only two layout breakpoints by default. When you create a new remote form in Studio 10, it will contain two initial layout breakpoints: 320 and 768. In previous versions, a layout breakpoint at 1024 was added to new forms but this has been removed. You can still add your own layout breakpoints, or change or delete the default values.

The $initiallayoutbreakpoints library preference has also been amended and the corresponding setting in the config.json file.

HTML template

A new property $htmltemplate has been added to remote task classes, allowing you to specify a different HTML template to use to test a remote form, rather than using the default template ‘jsctempl.htm’. For example, you may want to create a template with your own set of parameters in the “omnisobject” , but retain the default template.

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

PDF Printing

You can specify an alternative folder to place PDFs created in the JavaScript Client, rather than using the default “omnispdf” folder. There is a new item called "omnispdfFolder" in the "pdf" section of the Omnis configuration file (config.json) that allows you to specify the path of a folder to receive PDFs, overriding the default location. The new item defaults to empty, which means Omnis will use the current omnispdf folder. The folder specified in "omnispdfFolder" must already exist, otherwise Omnis reverts to the default omnispdf folder.

Remote Form Padding

A new property $layoutpadding has been added to responsive remote forms to allow you to set the amount of padding under the bottom-most control on the form. In previous versions, the bottom edge of the form was set to 2 pixels under the bottom-most control.

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

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

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

Message Dialogs

Header Styles

The appearance of various message boxes and dialogs has been improved including message dialogs created using $showmessage() or $clientcommand() and the commands 'yesnomessage', 'noyesmessage', and 'okcancelmessage'.

Title bars now have a larger touch area to make it easier to drag dialogs on touch devices. Title bar buttons are now larger, including larger high-resolution icons and better hover/focus characteristics. When hovering over a draggable area with the pointer, it changes to a grab pointer, and then to a grabbing pointer when the pointer button is held down to drag.

Dialog buttons have been based on the material design UI scheme which work from *right to left* instead of *left to right* and the primary button has a different color to the other buttons.

CSS classes have been added to dialog Headers so that different dialog types can have individual styling (errorheader, query header, etc.): the default values for these are in a new CSS file **omn_dlg.css.** The header and body styling of the dialogs can be changed in omn_dlg.css under .omnis-wf-title.typeheader and .typebody.

Javamessage Icons

The message dialog displayed using the 'javamessage' command (using $clientcommand) now contains a standard icon from the material design set; this specific change only applies to the javamessage, no other dialog boxes.

Types 'error', 'warning', 'success', 'prompt' and 'query' all contain an icon specific to that type, while 'message' does not use an icon. The images for the icons can be changed in omn_dlg.css under the .typeicon classes.

Managing Server Timeouts

You can now better manage what is displayed in the end user's browser when the Omnis Server responds with a Server error or Disconnected message. You can create a client-executed remote form method named $ondisconnected which will be called when there is an error on the server or the client is disconnected.

The method has a single parameter which provides the error text. This is only populated if it was triggered by a server error, rather than a disconnect due to Remote Task timeout etc. If you wish to prevent the default behavior, you must return kTrue from this method.

The form which initiated the server request will be queried for the method first. If it is not found, or it does not return kTrue, any parent forms (if it is a subform) will be tried.

Closing Browser Windows

A new method $closeurl() has been added that allows you to close a browser window that was previously opened by the $showurl() method. The $closeurl() method takes a single parameter, a string identifier to the window, returned in the fourth parameter of the $showurl() method.

JavaScript Components

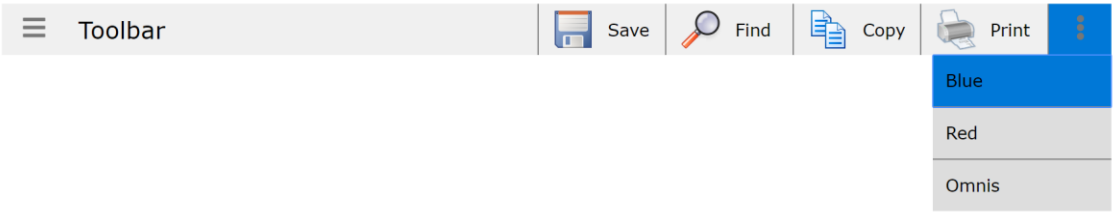
There are some new JavaScript controls, including a Toolbar control and an iCalendar external object (which can be used in remote forms and window classes), plus some of the existing JavaScript controls have been enhanced, including the Segmented Control, Progress bar Control, File Control (for uploading and downloading files) and the Data Grid control.

Toolbar Control

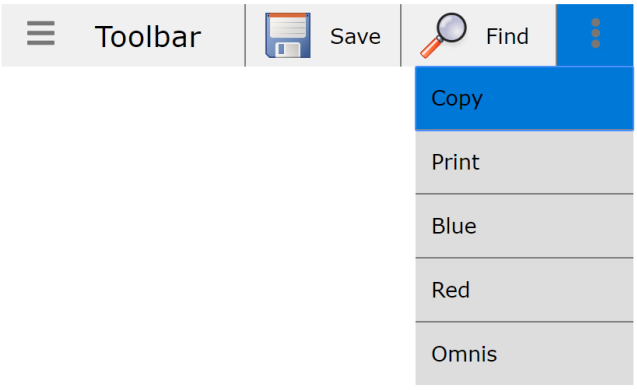
The **Toolbar Control** is a new JavaScript Control that can contain a number of buttons which the end user can click on or tap to perform an action. Each toolbar button can be assigned an icon and text, as well as a different action. When a button is clicked, the item number is reported to the event handling method allowing you to run the appropriate code.
A toolbar can have a side menu by setting $sidemenu to true and adding a list variable name to $dataname containing the menu items. Items are added to an overflow menu automatically (shown by three vertical dots) if they do not fit on the toolbar, or items can be forced to appear on the overflow menu. The toolbar is displayed horizontally, by default, but can also be displayed vertically. You can use edgefloat properties to ‘stick’ the toolbar to the top or side of the remote form.

The following example Toolbar has four items or buttons defined, each with an icon and text, a main title ‘Toolbar’ on the left, and an overflow menu on the right.

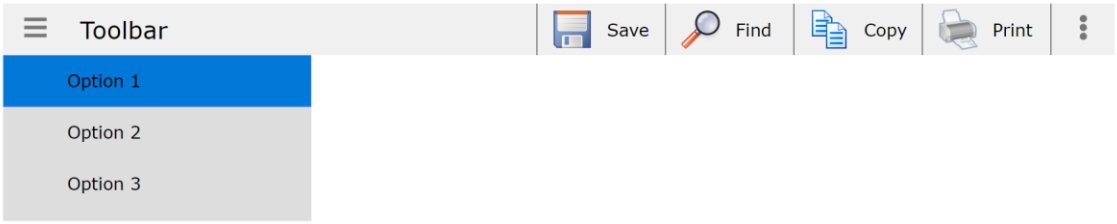
Items can be forced to always appear in the overflow, regardless of the width of the main toolbar, shown on the right of the toolbar by the three vertical dots, and shown dropped here:



As the browser window is resized, or the remote form (the app) is displayed on a mobile device, the main toolbar width will shrink, and the button items are added to the overflow menu automatically, as shown:



The following image shows the same Toolbar with the side menu added and in the dropped state:



There is an example app in the Hub to show how you can use the Toolbar control.

Properties

The custom properties for the Toolbar Control are described below. The 'Item' tab contains item specific properties that apply to the $currentitem.

Property	Description
$itemcount	The number of toolbar items/buttons
$currentitem	Item specific properties are assigned to the current item
$moveitem	Allows you to move an item in design mode; the current item moves to the position specified by the number entered
$itemiconid	The icon for the current item (item specific property), $showitemicons needs to be enabled to display icons; note that icons are not displayed on overflow items
$itemtext	The text for current item (item specific property); note $showitemtext needs to be enabled to display text
$itemoverflow	Force the item to be appear in the overflow menu (item specific property)
$sidemenu	Add a side menu to the toolbar. The $dataname must be a list containing the menu data
$dataname	Name of a list variable containing the menu data, to display a menu when $sidemenu is set to true
$verticaltoolbar	Display the toolbar in a vertical orientation
$menudirection	A kJSToolbarMenuDirection… constant which sets the direction the menu should open. Different values are available depending on the $verticaltoolbar property: Down or Up for horizontal toolbars, Right or Left for vertical toolbars
$toolbartitle	The optional title to display on the toolbar; leave this blank for no title
$titlefontsize	The font size applied to the toolbar title
$itemwidth	The width of items on the toolbar; without a value items have a variable width and are forced to fit the length of the toolbar
$displaystyle	A kJSToolbarStyle... constant determining the position of the icon text relative to the icon: either Above, Below, Left of, or Right of the icon

$showitemicons	If true, any item with an $itemiconid will display an icon on the toolbar; note that icons are not displayed on overflow items
$showitemtext	If true, any item with $itemtext will display text on the toolbar; when true, $showtooltips is disabled
$showtooltips	Show tooltips on toolbar items; $showitemtext must be set to false
$showdividers	If true, dividers will be shown between toolbar items
$dividercolor	The color of the dividers between items if $showdividers is true
$iconcolor	The color of standard icons such as the hamburger icon
$sidemenucolor	The background color of the side menu
$overflowcolor	The background color of the overflow menu
$toolbaractivecolor	The color of toolbar items when clicked
$toolbarhovercolor	Hover color for toolbar items
$sidemenuhovercolor	Hover color for side menu items
$overflowhovercolor	Hover color for overflow items
$selecteditem	The number of the selected item
$showselecteditem	If true, the item will have its background color set to $selectedcolor and its text colour set to $selectedtextcolor.
$selectedcolor	The background color of the selected item
$selectedtextcolor	The text color of the selected item

Clicking on a toolbar item will make that item selected, and $selecteditem is set to the selected item number. If $showselecteditem is true, the item will have its background color set to $selectedcolor and its text color set to $selectedtextcolor.

Events

The Toolbar reports two events: **evClick** reports the toolbar item that was clicked, with pClickedItem returning the item number; and **evNavigationClick** reports true if an item on the side menu was clicked, with the group number reported in pClickedMenuGroup (zero if the data is ungrouped) and the item number in pClickedMenuItem. You can write event handling code in the $event for the toolbar to trap these events and branch according to the value of pClickedItem or pClickedMenuItem.

Defining the $dataname list

To enable the side menu, you need to set $sidemenu to kTrue and specify a list variable name in $dataname containing the contents of the menu. The $dataname can generate either a grouped or an ungrouped menu.

Ungrouped list columns with each row representing an item:

- Text (Character): The text of the menu item.
- IconPath (Character): A URL of an image to display. The image will be scaled to fit.

Grouped list columns with each row representing a group:

- SubList (List): A list with columns matching the ungrouped list above. Contains data for the group.
- GroupName (Character): The text displayed on the group header.
- Fixed (Boolean): Optional column. If true, the group is always expanded. False by default.
- Collapsed (Boolean): Optional column. If true, the group is collapsed by default. False by default.

iCalendar External Component

iCalendar is a new, non-visual External Component that you can use in your Remote Forms (or window classes) to load and manage calendar events. iCalendar allows you to read, write and modify objects based on the standard iCalendar format, which is supported by many third-party calendar products.

The iCalendar model is based on four object types:

- **Component:** A group containing Properties which represent, for example, an event. Components can contain other Components (sub-components).
- **Property:** Used to communicate information about a Component, such as a description or a location.
- **Value:** Properties have a value associated with them. For example, a DTSTART Property will have a date or datetime value.
- **Parameter:** A modifier for a Property. Properties may have more than one Parameter (or none).

There are two types of object in the Omnis iCalendar external component:

- **Document:** Represents the entire Document and its children.
- **Component:** Used to access and manipulate iCalendar Components and their associated Properties and Parameters.

To access the iCalendar objects & methods, you need to create an instance variable in your remote form (or window class), choose Object as its Type, under Subtype drop down the Select object dialog, open the 'iCalendar Objects' group and select 'Document' object.

Working with iCalendar files

iCalendar Documents can be initialised with character data, or built up with the Omnis iCalendar methods.

To load an iCalendar file into a Document object, use FileOps to read in the character data. Then use $initwithdata() to initialise the document.

```
Do FileOps.$getfilename(lPath,"Select ics file","*.ics")
Do lFileOps.$openfile(lPath,kTrue)
Do lFileOps.$readcharacter(kUniTypeUTF8,iCalText)
Do lFileOps.$closefile()
Do lDoc.$initwithdata(iCalText)
```

To output the character data, use $getdata() on the Document or a single Component. To save the data into a file, use FileOps.

```
Calculate lDocText as iNewDoc.$getdata()
Do FileOps.$putfilename(lPath,,"*.ics")
Do lFileOps.$createfile(lPath)
Do lFileOps.$openfile(lPath)
Do lFileOps.$writecharacter(kUniTypeUTF8,lDocText)
Do lFileOps.$closefile()
```

Updating sub-components

The functions $getcomponent() and $getsubcomponent() return a copy of a Component. Therefore, modifying the returned Component will not affect the parent (the Component or Document that the method was called on). In order to update the parent, Component copy will need to be saved back to the parent with $replacerootcomponent() or $replacesubcomponent().

Custom Properties

As well as the standard Property types, custom Properties can be added to Components. These must be prefixed with “X-”, e.g. “X-PROPERTY”. By default, the data type of a custom Property is character. When adding a Property to a Component with $addproperty(), the optional iDataType Parameter can be used to override the default data type. Doing so will set the “VALUE” Parameter to the data type associated with the constant. The data type cannot be changed after it has been created.

Custom Parameters

Like custom Properties, custom Parameters can be applied to Properties. Similarly, they must also have “X-” as a prefix.

Error Properties

When a Document is initialised with $initwithdata(), the character data is parsed. If there are any syntactic or semantic errors in the data, such as a misspelt Property name or a Property without a value, an X-LIC- error Property will be inserted. For example, the following error is caused by misspelling the ATTENDEE Property:

X-LIC-ERROR;X-LIC-ERRORTYPE=PROPERTY-PARSE-ERROR:Parse error in property name: ATENDEE

Special Values

These values contain multiple parts and are therefore represented as rows. The static $createrow() helper method can be used to build these rows, which can be used to create a new Property or update a value.

Recur

The “RECUR” value type in the iCalendar model denotes a recurring event. It is commonly used with the “RRULE” Property. Its value may contain several parts, separated by semicolons. The parts contain key value pairs separated by the equals sign. The example shows a Recurrence Rule property.

RRULE:FREQ=MONTHLY;BYMONTHDAY=1;UNTIL=19980901T210000Z

A Component’s $propertylist will store an RRULE Property as a rows containing columns relating to each keyword. To create an RRULE Property with $addproperty(), the vValue parameter can take either a character or row argument. To create a recurrence type row, use $createrow(kICalendarRowTypeRecur).

Duration

The “DURATION” value type is represented in a component’s $propertylist as a row. The columns are: IS_NEGATIVE, DAYS, WEEKS, HOURS, MINUTES and SECONDS. To add a Property with a duration type, a row containing these column names can be used. The column values are all Integers, with the exception of IS_NEGATIVE, which is a Boolean. Alternatively, a string can passed. For example, P15DT5H0M20S denotes a duration of 15 days, 5 hours, and 20 seconds. See the RFC 5545 iCalendar specification for details on this format (https://tools.ietf.org/html/rfc5545#section-3.3.6).

Period

“PERIOD” value types have two parts: The first is the start time (date time). The second can either be the end of the period (date time), or a duration. Period is the default value type of the Free/Busy Property. In the $propertylist, period values are

displayed as a row containing a START date time and either a DURATION or an END date time.

19970101T180000Z/19970102T070000Z	date time "/" date time (Explicit)
19970101T180000Z/PT5H30M	date time "/" duration (Start)

Geo

"GEO" Properties hold geographic coordinates, represented as two floats separated by a semicolon. The values are latitude and longitude respectively, e.g. 37.386013;-122.082932.

In the $propertylist of a Component, they are displayed as a row containing a LAT and a LONG column with float values.

Methods & Properties

Static Methods

$createcomponent()

OmnisICalendar.$createcomponent(iType)

Creates a new iCalendar Component object using one of the kICalendarComponent... constants. Returns an iCalendar component object.

- iType: A kICalendarComponent... constant to specify the Component type.

$createrow()

OmnisICalendar.$createrow(iType)

Returns a row which can be used to add or update certain Properties. iType can be one of the kICalendarRowType... constants.

- iType: A kICalendarRowType... constant to specify the type of row.

Document Object

Methods

$initwithdata()

$initwithdata(cData)

Initializes the object with the Character contents of an iCalendar file. Returns true if successful.

- cData: The character data containing the contents of an iCalendar file.

$getdata()

$getdata() - no parameters

Returns character data representing the Document that can be saved as an iCalendar file.

$getcomponent()

$getcomponent(iComponentId)

Returns a copy of the root Component object with the specified ID.

- iComponentId: The ID of the root Component to find.

$addrootcomponent()

$addrootcomponent(oComponent)

Adds a Component to the root of the Document. Returns the ID of the new Component.

- oComponent: The Component to be added to the root.

$deleterootcomponent()

$deleterootcomponent(iComponentId)

Removes the Component with the specified ID from the root. Returns true if the Component was deleted.

- iComponentId: The ID of the Component to delete.

$replacerootcomponent()

$replacerootcomponent(iComponentId, oComponent)

Replaces the Component at the specified ID with oComponent. Returns true if successful.

- iComponentId: The ID of the Component to replace.
- oComponent: The new Component.

Properties

$componentlist

A list of root-level Component info for the Document. The columns for this list are: ID, Type and TypeName.

Component Object

Methods

$getdata()

$getdata() - no parameters

Returns character data representing the Component and its children, that can be saved as an iCalendar file.

$isvalidcalendar()

$isvalidcalendar() - no parameters

Returns true if the VCALENDAR Component meets the RFC 5546 iCalendar specification standards.

$getsubcomponent()

$getsubcomponent(iComponentId)

Returns a copy of the sub-component object with the specified ID.

- iComponentId: The ID of the sub-component to find.

$addsubcomponent()

$addsubcomponent(oComponent)

Adds a sub-component to the Component. Returns the ID of the new Component.

- oComponent: The sub-component to be added to the Component.

$deletesubcomponent()

$deletesubcomponent(iComponentId)

Removes the sub-component with the specified ID from the component. Returns true if the sub-component was deleted.

- iComponentId: The ID of the Component to delete.

$replacesubcomponent()

$replacesubcomponent(iComponentId, oComponent)

Replaces the Component at the specified ID with oComponent. Returns true if successful.

- iComponentId: The ID of the Component to replace.
- oComponent: The new Component.

$addproperty()

$addproperty(cName, vValue, [wParameters, iDataType])

Adds a new Property to the Component. Returns the Property ID.

- cName: The name of the Property. Must be a valid Property type.
- vValue: The value to assign to the Property. The type can be Character, Integer, Date Time, Float, Boolean or Row.
- wParameters: A row of Parameters to add to the Property.

- iDataType: A kICalendarDataType... constant. Sets the 'VALUE' Parameter which overrides the default data type for the Property. Can be used to specify the type of a custom Property.

$deleteproperty()

$deleteproperty(iPropertyId)

Delete the Property with the specified ID. Returns true if the Property was deleted.

- iPropertyId: The ID of the Property to delete.

$setparameter()

$setparameter(iPropertyId, cName, cValue)

Sets the Parameter of the Property. If there is an existing Parameter with the same name, it will be overwritten. Returns true if successful.

- iPropertyId: The ID of the Property associated with the Parameter.
- cName: The name of the Parameter to set.
- cValue: The value to set the Parameter to.

$updateproperty()

$updateproperty(iPropertyId, vValue, [wParameters])

Updates the Property with the specified ID. Providing Parameters will overwrite any existing ones. Returns true if successful.

- iPropertyId: The ID of the Property to update.
- vValue: The new value to assign to the Property. The type can be Character, Integer, Date Time, Float, Boolean or Row.
- wParameters: A row of Parameters to add to the Property.

$deleteparameter()

$deleteparameter(iPropertyId, cName)

Removes the Parameter with the name cParamName from the Property. Returns true if the Parameter was deleted.

- iPropertyId: The ID of the Property associated with the Parameter.
- cName: The name of the Parameter to delete.

Properties

$componentlist

A list of sub-component info for the Component. The columns for this list are: ID, Type and TypeName.

$propertylist

A list of iCalendar Properties held by this Component. The columns for this list are: ID, PropertyName and PropertyValue. The PropertyValue column contains a row for each Property. Each row has a "_VALUE" column containing the Property value (the type of this depends on the Property), and columns for any Parameters that the Property has.

$typename

The type name of the Component.

$typenumber

The type number of the Component.

Edit Control

Shortcut Keys

Various shortcut keys have been added to Edit controls to allow you to select text and move the insertion point within a standard JavaScript Edit Control (the new shortcuts also apply to Window class Entry fields).

The shortcut keys are stored in a new Omnis preference **$keys** which can be edited in the Property Manager. The shortcut keys for Edit controls are stored in a new configuration file called 'keys.json' and located in the Studio folder (this is the same file containing the new shortcuts for the Method Editor). The file is created the first time you edit the shortcuts in the Property Manager and click OK.

Shortcut key	Action
Alt+End	End of Text Alternative
Alt+Home	Start of Text Alternative
Ctrl+Alt+DownArrow	Scroll Down
Ctrl+Alt+LeftArrow	Scroll Left
Ctrl+Alt+RightArrow	Scroll Right
Ctrl+Alt+UpArrow	Scroll Up
Ctrl+DownArrow	Paragraph Down
Ctrl+End	End of Text
Ctrl+Home	Start of Text
Ctrl+LeftArrow	Backwards Word
Ctrl+RightArrow	Forwards Word
Ctrl+UpArrow	Paragraph Up
End	End of Line
Home	Start of Line
PageDown	Page Down
PageUp	Page Up

Content Selection

A new method **$setselection** has been added to the Edit control to allow you to select a range of characters within the control.

- $setselection(iFirstSel[,iLastsel]))
  Sets the focus on and selection range of the content in the Edit control. If iLastSel is omitted selection will occur to the end of the edit field. Returns the selected text.

$setselection has two parameters, both Integer, iFirstSel and iLastSel to set position of the characters to be selected within the edit control. iLastSel selects up to, *but not including,* the character specified, and if omitted the content in the edit field is selected to the end. The method returns the content selected.

Horizontal Padding

The property **$horzpadding** has been added to the Edit control to allow you to add extra horizontal padding, in pixels, inside the control; when applied this property adds padding on the left and right of the text within the edit control.

Multiline Edit Scrolling

Text wrapping for the JavaScript Multiline Edit field is now prevented if the $horzscroll property is enabled (kTrue). However, if $autoscroll is true, then text wrapping does occur (since $autoscroll is on by default).

Segmented Control

A number of properties have been added to the Segmented Control to allow you to control its appearance, such as the ability to add space between the segments (buttons) or to add rounded corners.

Segment size and spacing

The following new properties control the segment size, spacing and appearance: $segmentspacing, $segmentwidth, $segmenteffect, $segmentbordercolor and $segmentborderradius.

- **$segmentspacing**
  the space between the segments in pixels. The behavior can be affected by $segmentwidth (see below). If zero, dividers are drawn between segments. Otherwise borders are drawn around the segments.
- **$segmentwidth**
  The width applied to all the segments in pixels. By default, this is zero, in which case the width of the segments is determined by the total width of the control and $segmentspacing. If this value is small enough, the segments will be centred in the control.

If the $segmentwidth and the $segmentspacing are set such that the segments extend beyond the width of the control, the overflowing content will be scrollable. However, if $segmentwidth is zero (the default), the segments will always fit inside the container.

In the extreme case where $segmentspacing is very high, as long as $segmentwidth is zero, the spacing will be limited to prevent the segments becoming too small or the content overflowing.

- **$segmenteffect**
  Determines whether borders / dividers are applied to segments, either kBorderNone or kBorderPlain
- **$segmentbordercolor**
  The colour that applies to borders / dividers of segments.
- **$segmentborderradius**
  Single value border radius that applies to segments. If $segmentspacing is zero, this applies to only the outer edges of the outer segments. Otherwise it applies to all segments.

Hiding Disabled Segments

You can set $segmentenabled for a segment to false to disable it. The new property $hidedisabledsegments allows you to hide any segments that have been disabled.

Moving Segments in Design mode

There is a new design-time property $movesegment that lets you move a segment: you need to set it to a number corresponding to the new position (the property works in the same way as the Data Grid's $movecolumn property).

Progress Bar Control

The **Progress Bar** control has some new properties to improve the appearance of the progress bar on all platforms: **$usesystemappearance, $secondarycolor** and **$progressanimation.**

- **$usesystemappearance**
  (boolean, true by default) If true, the progress control uses the <progress> HTML5

element (as long as it's supported by the browser).
If false, the progress control is built with two elements

- **$secondarycolor**
  Sets the colour of the stripes of the progress bar. Only applies when $usesystemappearance is false.
- **$progressanimation**
  (boolean, true by default) Animates the progress bar. Only applies when $usesystemappearance is false.

File Control

The File control has had a number of enhancements to improve its usability and appearance for uploading and downloading files in the JavaScript Client.

Uploading Multiple Files

The File control now allows multiple files to be selected for uploading by setting $allowmultiple to kTrue. It is not supported by some browsers, just like $maxfileuploadsize (IE9 and below, Opera).

The properties $maxbatchuploadsize and $maxbatchuploadsizeerrortext have been added to work similarly to $maxfileuploadsize to impose a limit of the total amount of data to be uploaded. This works independently of $maxfileuploadsize so you can impose a limit on single or multiple file uploads. In addition, $uploadedprogresstextbatch has been added to show the progress of the batch of files, and works similarly to $uploadprogresstext.

Error messages shown when file sizes are exceeded, for single or batches of files, now give the user feedback on what the size limits are and lists the offending files exceeding the limit.

The UI for the File control has also been improved to provide better formatted information. On single upload dialogs the file name is displayed above the progress bar, to the left, the percentage is shown above to the right, and file upload size information shown below as before. Multiple file upload dialogs display the same information for each single file, while another progress bar shows progress through the batch of files, including how many files have uploaded out of the total.

File sizes displayed to the user are now in a more readable unit so when 1000 bytes is exceeded it changes to kB, 1000 kB changes to MB and 1000 MB changes to GB.

Upload File Type

The File control has a new property $uploadtypes which allows you to filter the file types that can be uploaded. The property accepts a comma-separated list of file extensions or MIME types, for example, the string '.png, .jpg, .jpeg' would allow PNG or JPG files, or 'image/*' to allow any image files.

Localization

All the text and labels in the File control can now be translated via the jOmnisStrings object in the JavaScript client. See the main Localization section for more info.

Pie/Bar Chart

You can specify your own colors for Pie and Bar Charts: in previous versions you had no control over the colors displayed in charts. There is a new runtime-only property $colorlist for Pie and Bar charts that allows you to specify a list of colors to use for the segments or bars in the chart.

You need to create a list of strings representing CSS colors and assign the list to the $colorlist property, for example:

```
Do iColorList.$define(iColor)
Do iColorList.$add("#CE3D3D")
Do iColorList.$add("rgb(81, 206, 61)")
```

```
Do iColorList.$add("hsl(230, 60%, 52%)")
Do iColorList.$add("Gold")
Do $cinst.$objs.PieChart.$colorlist.$assign(iColorList)
```

The accepted color formats are: Hex Code RGB, Decimal Code RGB, HSL, or Color Name, and the formats can be mixed throughout the list as in the example above.

If there are not enough colors available in the color list for the number of segments in the chart, then Omnis will repeat the colors in $colorlist. Therefore, if you want to avoid repeating colors, create a color list containing more colors than you will generally need to cater to the number of data points in your chart.

Data Grid

Validating data

Data grids have a new client-executed method, $validate, which allows you to validate the data entered into any cell in the grid. If present, the method is called when an edit is made to a grid cell, with the parameters pRow, pCol, pNewValue being passed to the method. The method returns true to indicate that the change is valid, depending on the validation code you add to the method, otherwise the value in the cell will revert to the previous value.

Copying data

Data Grids (and standard list controls) now allow the end user to copy data from selected rows. Data grids return the copied rows as tab-separated values. You can add a client executed method named "$clipboardcopy" to the control to handle the clipboard content. The method can return character data or a list. If it is a list, column 1 must be the MIME type and column 2 must be the content.

For example:

```
# $clipboardcopy client method
Do lList.$define(lMime,lContent)
Do lList.$add("text/plain","Copy this as plain text")
Do lList.$add("text/html","Copy this as <b><u>HTML</u></b> instead")
Quit method lList
```

evCellValueChanged event

There is a new event for Data Grids, evCellValueChanged, to report when the user has changed the *value* of a cell, while there has been a small change to the existing evCellChanged event.

- **evCellValueChanged** (pHorzCell, pVertCell)
  sent when the user has changed the value of a cell.
  pHorzCell - The column number of the cell that has changed.
  pVertCell - The row number of the cell that has changed.
- **evCellChanged** (pHorzCell, pVertCell)
  sent when the current cell has changed, e.g. when navigating between cells with the arrow keys or clicking a cell that isn't the current cell.
  pHorzCell - The column number of the new current cell.
  pVertCell - The row number of the new current cell.

Fixed Columns

Data grids have a new property $frozencolumns which allows you to fix or "freeze" a number of columns to the left of the grid, so they do not scroll when the other columns in the grid are scrolled horizontally. The property takes a number value from 1 upwards corresponding to the first column on the left of the grid. For example, you could specify a value of 1 to create row headings that are fixed to the left of the grid.

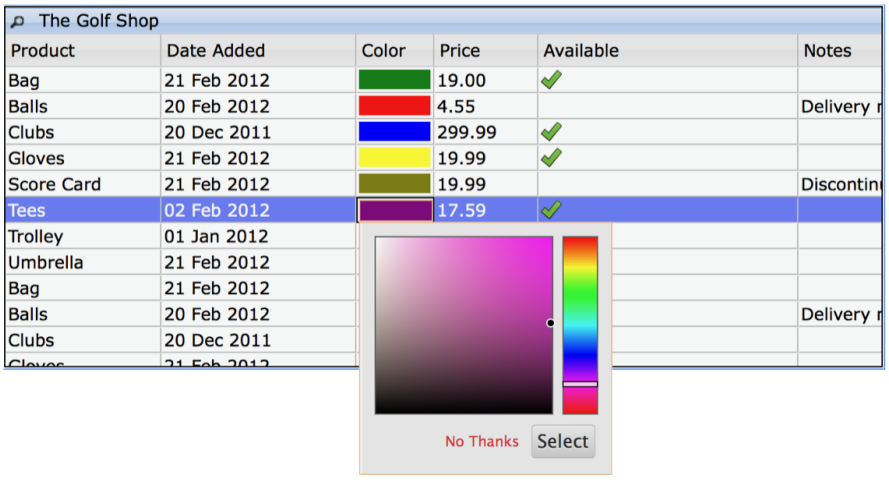
Color Picker

Data grid columns have a new column type kJSDataGridModeColorPicker which means the column will display a color picker allowing the end user to select a color. A

numeric color value is returned from the picker, or a color functions can be used to set the color of the column, such as truergb(kDarkGreen), or rgb(255,0,0).

For example, to set the colors for the first 3 lines in the third column, use the code:

```
Do iList.$add('Bag','21/02/12',truergb(kDarkGreen),'19.00',kTrue,'')
Do iList.$add('Balls','20/02/12',rgb(255,0,0),'4.55',kFalse,'Delivery next
 week')
Do iList.$add('Clubs','20/12/11',rgb(0,0,255),'299.99',kTrue,'')
```



You can specify the text for the OK and Cancel buttons on the color picker using $colorpickeroktext and $colorpickercanceltext.

An entry field has been added to the color picker which accepts colors in the hex (the default), rgb or color name formats. Localisable strings have been added for the color entry field for the aria-label and aria-describedby accessibility properties, "ctl_dgrd_color_input" and "ctl_dgrd_color_input_desc" respectively. In addition, end users should be able to navigate the color picker from the keyboard without the picker losing the focus.

Number Columns

Data grid columns with type Number have a new property $columnzeroshownempty which specifies that values of zero are shown empty rather than displaying a 0 digit.

Hiding a column

Data Grids have a new property $columnhidden which allows you to hide the specified column at runtime. The default is false, meaning the column is visible.

Rich Text Editor

The Rich Text Editor control now allows you to print the text contents of the control. There is a new print button on the editor's toolbar, which when clicked opens a window for printing the contents of the editor. There is also new method $printcontents in the control which you can use to print the contents.

- **$printcontents**(cTitle)
  Opens a new window to print the editor's current contents. cTitle is the title of the document to print.

You enable the new print button by setting $removedtoolbaritems to kJSRichTextPrint. In addition, there is a new Omnis string table item with ID: rt_print which you can edit to change the tooltip of the button.

Lists

Changing Current Line from the Keyboard

A new property, $keyboardchangesline, has been added to JavaScript List controls, including standard Lists, Native lists, Tree lists, and Data grids.

When set to true, navigating the list with the keyboard *also sets the current line,* so when the list is not in a multiple select state, there is not a separate focused line. The evClick event is fired when the current line changes.

When set to false, navigating the list with the keyboard always uses a focus line and the user has to manually select a current line with the Space or Enter key, at which point evClick is fired.

Subforms

Error text

The $errortext property is only supported for subform controls when they are not scrollable, i.e. when $vertscroll & $horzscroll are both kFalse and the subform class is not responsive.

Subform Sets

Title Bar Appearance

The appearance of the title bar for subforms in a subform set has been improved. Specifically, the close, minimize and expand buttons have been replaced with larger icons and the hover behavior has been improved.

Positioning

When a subform is added to a Subform Set (using the subformset_add client command), and formLeft or formTop parameters are set to kSFScenter, the subform will be displayed in the center of the current viewport or the current form, whichever is the smaller of the two: note that horizontal and vertical centring work independently of each other. This improves on the previous behavior of forcing the browser to scroll to the centre of the main form, to center the new subform, which could be a long way from the current view on large forms.

Nav Bar

There is a new property, $disableanimation, added to the JavaScript Nav Bar control. The new property was added to fix a problem when using the built-in VoiceOver screen reader on an iPad in conjunction with a Nav Bar linked to Page Pane: the problem is avoided by disabling the animation effect on the Nav Bar.

The $disableanimation disables the animation when moving between pages. This property can only be set in design mode (not at runtime using the notation).

HTML Object

The HTML Object has a new property $wraptext to allow the content in the control to wrap.

- $wraptext
  If true, the content in the control will wrap. It is true by default.

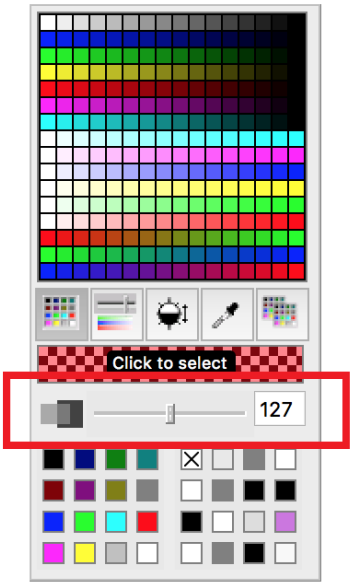
This property sets white-space for the control to 'normal' if true, and 'no-wrap' if false. Therefore, this may change the behavior if the HTML control is in a paged pane, and possibly in other cases where the white-space would have been inherited by a parent element.

Page Pane

A default CSS classname 'omnis-pagedpane-page' has been added to the Page Pane control. This allows you to apply CSS styling or behavior to each page of the paged pane control.

In addition, a CSS rule (-webkit-overflow-scrolling: touch;) has been added to enable momentum scrolling on iOS, i.e. for touch iOS devices, scrolling slows down before stopping.

Alpha Colors for Controls



as **rgba(255,0,0,127)**

Some of the Remote form class controls and Window class controls now support alpha colors, meaning that you can set the transparency for the color of the control. The color selection dialog in the Property Manager will now display an alpha selection slider if the current selected control supports alpha colors (the alpha slider is hidden for controls that do not support alpha).

The controls that support alpha colors include the Line, Oval, Rect and RoundRect background objects for windows.

There is a new function ***rgba()*** that can be used to set the RGB color and alpha setting for controls. The syntax is rgba(red,green,blue,alpha) with each parameter being an integer value in the range 0-255, where an alpha value of 255 means completely transparent. For example, to set the color of a window background object, in this case 50% transparent red:

```
Calculate $cwind.$bobjs.1016.$forecolor
```

In addition, the color selection palette for the controls that support the use of a color palette or a popup color palette, including the colorpalette control, push buttons, and toolbars, now include the new alpha selection slider.

When assigned a color with no alpha the palette will automatically hide the alpha slider. If the control is assigned an alpha value, the palette will display the new alpha slider.

For example, where **colorbutton** is a push button with $buttonmode set as kBMcolorpicker:

```
Do $cwind.$objs.colorbutton.$contents.$assign(rgb(255,0,0))  ## will cause the
   color palette not not show an alpha value
Do $cwind.$objs.colorbutton.$contents.$assign(rgba(255,0,0,127))  ## will
   cause the color palette to show an alpha value
```

The Omnis external component interface has been updated so external components can support alpha colors. The example WASH control has been updated to demonstrate this.

The Export & Import Library to JSON options have been updated to support alpha color values for controls.

Tree List

Tree lists now scroll to view the current line in the list, and any parents opened as necessary, whenever the current line changes or the $currentnodeident property is set. In a non-multiple select tree lists, setting either the current line in the list or the $currentnodeident will select the line and scroll to it. This applies to flat list trees only.

In multiple select tree lists, setting the current line will scroll to that line but will not select it. Setting $currentnodeident notationally will set both the current line and select the node. Tree lists without checkboxes will clear any current selection, but tree lists with checkboxes will not. This behavior mimics the user behaviour of clicking a node.

In multiple select tree lists, the $currentnodeident and the current line in the list can reference different nodes; however, in single select tree lists, they will always reference the same nodes.

Droplists

Horizontal Padding

The Droplist and Combo Box controls now have the property $horzpadding to allow you to add extra horizontal padding, in pixels, to the text in the list part of the control. This property has also been added to Combo boxes.

Selected Value

A new property $seldataname has been added to the Droplist control that allows you to specify the name of an instance variable, which will be populated automatically with the selected value from the droplist.

$active Property

As part of the work to make the JavaScript Client meet Accessibility guidelines, a new *$active* property has been added to all JavaScript controls; the $active property is set to kTrue for all new controls, except the Label Control which has $active set to kFalse. The $active property allows you to control whether a component is *active* (kTrue) or *inactive* (kFalse) – in an inactive state, a component *cannot be interacted with at all,* so the end user cannot tab to it, the contents cannot be selected or scrolled (in a list), and user clicks on an inactive control are ignored. Therefore, when a control is inactive, it is completely ignored in the tabbing order, so when the end user tabs the focus will jump to the next active control – in the context of accessibility, an inactive component will be ignored.

$enabled property

Most JavaScript controls had the $enabled property in previous versions which allowed you to disable or enable the control. For some controls, the $enabled property has been removed and replaced with the $active property. Other controls have kept the $enabled property, but also have the new $active property.

The behavior of the *$enabled* property is now better defined, so when false, the control is in a *read-only* state, which means it can be tabbed to, and interacted with in such ways that do not alter its state. For example, the end user can press tab to move the *focus* between lines in a list without changing the current line in the list.

The following table summarizes the presence and default setting of the $enabled property for JS controls in Studio 8.x, and the default setting or status of $enabled and $active for new JS controls in Studio 10.

	Studio 8.x	**Studio 10**	
JS control	**$enabled**	**$enabled**	**$active**
Activity Control	kTrue	Removed	kTrue
Background Control	kTrue	Removed	kTrue
BarChart Control	kTrue	kTrue	kTrue
Button Control	kTrue	Removed	kTrue
Checkbox Control	kTrue	Removed	kTrue
ComboBox	kTrue	kTrue	kTrue
Complex Grid	kTrue	kTrue	kTrue
Data grid Control	kTrue	kTrue	kTrue

	Studio 8.x	Studio 10	
JS control	**$enabled**	**$enabled**	**$active**
Date Picker Control	kTrue	kTrue	kTrue
Device Control	kTrue	kTrue	kTrue
Droplist	kTrue	Removed	kTrue
Edit Control	kTrue	kTrue	kTrue
File Control	kTrue	Removed	kTrue
HTML Object	NA	NA	kTrue
Hyperlink Control	kTrue	Removed	kTrue
Label Object	kFalse	kFalse	kFalse
List Control	kTrue	kTrue	kTrue
Map Control	kTrue	kTrue	kTrue
Navigation Bar Control	NA	NA	kTrue
Navigation Menu Object	kTrue	Removed	kTrue
Page Control	kTrue	Removed	kTrue
Paged Pane	kTrue	kTrue	kTrue
Picture Control	kTrue	Removed	kTrue
PieChart Control	kTrue	kTrue	kTrue
Popup Menu Control	kTrue	Removed	kTrue
Progress Bar Control	NA	NA	kTrue
RadioGroup Control	kTrue	Removed	kTrue
Rich Text Editor Control	kTrue	kTrue	kTrue
Segmented Control	kTrue	Removed	kTrue
Slider Control	kTrue	Removed	kTrue
Subform	kTrue	kTrue	kTrue
Switch Control	kTrue	Removed	kTrue
Tab Control	kTrue	Removed	kTrue
Timer Control	kTrue	Removed	kTrue
Toolbar Control	NA	NA	kTrue
TransButton Control	kTrue	Removed	kTrue

	Studio 8.x	Studio 10	
JS control	**$enabled**	**$enabled**	**$active**
Tree Control	kTrue	kTrue	kTrue
Video Control	kTrue	Removed	kTrue

Conversion

When a library is converted, controls that now only have an $active property will inherit the kTrue/kFalse value of the $enabled property. Any code that was used to assign $enabled on these controls at runtime will be automatically re-routed to apply to $active. This means that your applications should still work in the same way as before, but you are advised to thoroughly test the behavior of any controls that previously used the $enabled property, and any code that checked the value of $enabled.

On conversion, controls that previously had $enabled set to kFalse, that now have *both $enabled and $active properties,* will still have $enabled set to kFalse, but $active will be set to kTrue. An exception to this is if a Label control had $enabled set to kTrue, it will keep $enabled set to kTrue, but $active will be set to kFalse.

Context Menus

Context menus previously only opened via clicks onto a control if $enabled for the control was kTrue. In Studio 10, context menus are opened if $active of the control is kTrue. If you wish to disable this behavior for a control, you should use Quit event handler (discard event) when handling evOpenContextMenu in your event handling methods for the control.

Sizing Objects

You can use the **Same Width/Height** options on the **Align** menu to make all the fields on a form the same width or height using the *largest size* of all the currently selected objects. Now you can hold down the Ctrl (Windows) or Cmnd (macOS) key while selecting the menu item to use the *smallest size* in the group to set the width or height of the objects (also applies to window & report class objects).

Tooltips and Carriage Return

Text in tooltips will now wrap if it contains a carriage return character or other wrapping characters when the text width of the tip would exceed a third of the screen width. In previous versions, tooltips only used CR to line wrap when the width of the tip was greater than half the screen width.

In addition, the CR character is no longer displayed. However, any other control characters (characters less than space, or the character 0x7f) are displayed using the Unicode control character page.

This change also applies to tooltips for window class controls.

Adding Customized JavaScript Components

You can now add your own customized JavaScript components to the Component Store under your own tab. To do this, you need to create a new remote form, copy any components you want to customize from the JSFormComponents form, and add them to your own form. This might be useful if you always want to create edit controls or buttons with certain properties (e.g. colors or fonts)

You will need to edit the Component Store library (comps.lbs) to change the contents of the Component Store. To open the Component Store library, right click on the background of the Component Store itself and select 'Show Component Library in Browser'. We recommend that you do not change the components in the JSFormComponents form since these are the default components that appear in the

Component Store, rather you should create your own customized components using the following method.

Add a new remote form class to the Component Store library; note that the name of the new remote form will be used as the tab name in the Component Store toolbar. Set the $componenttype property of the remote form to kCompStoreDesignObjects using the Notation Inspector: to do this, open the Notation Inspector, click on the Search button (the cursor changes to a spy glass), click on your remote form in the Studio Browser, and in the Property Manager set $componenttype to kCompStoreDesignObjects (note the $componenttype property will only be displayed via the Notation Inspector).

With your new remote form open, open the JSFormComponents remote form next to it. Drag any JavaScript controls you want to customize from JSFormComponents into your remote form and change their properties or appearance as required. After you hide the Component Store library, the customized JavaScript controls will be available in the new group in the Component Store.

JavaScript Component Templates

When you add a JavaScript Component to a remote form in your code at runtime, Omnis 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.

Remote Debugger

Remote debugging allows you to debug your Omnis code remotely over the network. You use a development copy of Omnis Studio, the *remote debug client,* to connect over the network to another copy of Omnis Studio, the *remote debug server*. References to “the debugger” in this section refer to the remote debugger, while any references to the local Omnis Studio debugger use the term *local debugger*. Some key points to note:

- The remote debug server runs the code that is to be debugged, and it can be any type of installation: development, fat client runtime, server or headless server.
- Omnis code running in the multi-threaded server, in server stacks other than the main stack, can be debugged.
- The remote debug server and client do not need to be running on the same operating system.
- The version of the client must be the same or later than the version of the server.
- Protected classes and locked libraries can be debugged.

Although the term remote debugger is used, the remote debugger client and server can be on the same computer, and in fact the client and server can be the same Omnis process. In the latter case, the remote debug client runs with some restrictions, which are discussed later.

Connectivity

The remote debug client and server always connect to each other over a **WebSocket.** This applies even if the client and server are running in the same Omnis process. The WebSocket connection is a direct connection from client to server, so it may require a port in the firewall to be opened on the server. As WebSocket connections start as HTTP connections, a WebSocket can be a secure TLS connection, and it can require a

client certificate to authenticate the client. For the Remote Debugger, a TLS connection is always required, so the WebSocket starts as HTTPS.

An established connection between a remote debug client and server is called *a remote debug session,* or just *session*. A copy of Omnis that is running as a remote debug client or server, or both, can run *only one session* at a time.

Remote Debug Server

In the developer version of Omnis, you can configure the Remote Debug Server by clicking on the Omnis Studio (root) node of the Studio Browser tree, and clicking on the Remote Debug Server link.

In a runtime version of Omnis (not headless), if the library remotedebug.lbs is in the startup folder, there is a menu named Remote Debug. This contains a single menu item that can be used to open the window. In the headless server you can configure remote debugging via the OS Admin window.

The Remote Debug Server window has two tabs, one to control the server, and the other to configure the server.

The Control Server tab has a single button, used to start or stop the server - the button text changes depending on the current state of the server. Until the remote debug server is started, it will not accept a connection from a remote debug client.

The Configure Server tab allows you to enter configuration details for the remote debugger server. The Configure Server tab shows fields that correspond to the entries in the configuration file. These fields are described in the following sections.

Remote Debug Server Configuration file

The *remote debug server* configuration is stored in the file called remote_debug_server_config.json, located in the folder clientserver/server/remotedebug in the data folder of the Omnis installed tree.

You can edit this JSON file directly as a text file, or use the Remote Debug Server window, as described above.

You should note that Omnis uses a node.js server running alongside Omnis to provide the WebSocket server; this communicates with Omnis using a local in-memory socket. As a consequence, some of the configuration information stored in remote_debug_server_config.json is used by node.js. An example configuration file:

```
{
    "debugPort": 8080,
    "serverPfx": "server.pfx",
    "pfxPassPhrase": "xxxxxx",
    "ca": [ "server_cert.pem" ],
    "requestCert": false,
    "rejectUnauthorized": false,
    "userName": "myUser",
    "hashedPassword":
     "AAGGoAAAABBSEkknQUIeHQHu1sIyWxlSAAAAIHw9kvCVF4tE//SMpbSGVD/RKJLekoR7TlTv
  ZVy3MbkJ",
    "startRemoteDebugServerAtStartup": true,
    "pauseAtStartupUntilDebuggerClientStartsExecution": false,
    "logConnectionSetup": false
}
```

Debug Port

The TCP/IP port on which the WebSocket server listens for incoming connections from a client.

Server PFX

This is a file containing the server certificate and private key. It must be in the same directory as remote_debug_server_config.json. The default install tree has a self-signed certificate and key generated by the openssl command (available on any system where openssl is installed). You will need to provide your own private key and certificate. You can generate a new private key and self-signed certificate using the following openssl commands:

```
openssl req -x509 -newkey rsa:4096 -keyout server_key.pem -out server_cert.pem
   -nodes -days 1024 -subj "/CN=localhost/O=Demo" -passin pass:xxxxxx


openssl pkcs12 -export -out server.pfx -inkey server_key.pem -in
   server_cert.pem
```

This file is set as the pfx option when calling the node.js method https.createServer(). You can find more documentation about this in the node.js documentation online:

https://nodejs.org/docs/latest-v8.x/api/https.html#https_class_https_server
https://nodejs.org/docs/latest-
   v8.x/api/tls.html#tls_tls_createsecurecontext_options

PFX Pass Phrase

This is the pass phrase used to protect the Server PFX file. In the example in the previous section this is xxxxxx.

CA

See https://nodejs.org/docs/latest-v8.x/api/tls.html#tls_tls_createsecurecontext_options for more details. You would typically only set the CA when using a self-signed certificate, in which case it has a single entry. In the Server PFX section above, the certificate was signed using server_cert.pem. The general value of this is a comma-separated list of trusted CA certificate file names. The files must all be in the same directory as remote_debug_server_config.json.

Request Client Certificate

A Boolean option. If true, the node.js server requests a client certificate to authenticate the client. The client certificates are discussed later, in the client connectivity section.

Reject Unauthorized

A Boolean option. If true, the server will reject any connection which is not authorized with the list of supplied CAs. This option only has an effect if Request Client Certificate is true.

User Name

If not empty, the WebSocket connection also uses HTTP basic authentication to authenticate the user, in which case this field contains the user name used for HTTP basic authentication.

Hashed Password

If the User Name is not empty, this is the PBKDF2 hash of the password required for HTTP basic authentication.

Start Remote Debug Server

This Boolean option controls whether the remote debug server automatically starts when Omnis starts.

Pause Execution At Startup

If the remote debug server is configured to automatically start when Omnis starts, you can set this Boolean option to true to make Omnis pause execution at startup before it runs the startup tasks of libraries in the startup folder.

When using this option, Omnis displays a working message (Waiting for remote debug client to start execution…), and enters a loop where it waits for a remote debug client to open a session. Once a session is opened, Omnis remains in the loop, where it is now waiting for a command from the client to start execution. During this loop, the client can inspect remotely debuggable code, and set breakpoints for example.

The loop terminates either when the client sends a command to run startup, or when the remote debug session closes, or when a user clicks the cancel button on the working message displayed on the server. When the loop terminates, Omnis runs the startup tasks for the libraries in the startup folder.

Remote Debug Client

The remote debug client is accessible via a new node in the Studio Browser tree, "Remote Debug Client". It uses a similar session model to the Omnis VCS. When you click on the Remote Debug Client node in the tree, hyperlinks appear in the browser panel for Session Manager, and Open Session.

The session manager allows you to configure remote debug sessions. Each session provides the parameters that allow the remote debug client to establish a WebSocket connection to a remote debug server. These parameters are described in the following sections.

Name

A name that identifies the session.

Server

The IP address or DNS name of the remote debug server.

Debug Port

The debug port configured for the remote debug server. When connecting to the server, the client connects to a URL of the form

wss://Server:DebugPort

Client Certificate

If the server requires a client certificate, you specify this here.

You can generate a client certificate using the openssl commands:

```
openssl req -newkey rsa:4096 -keyout client_key.pem -out client_csr.pem -nodes
   -days 1024  -subj "/CN=192.168.1.11" -passin pass:xxxxxx
```

```
openssl x509 -req -in client_csr.pem -CA server_cert.pem -CAkey server_key.pem
   -out client_cert.pem -set_serial 01 -days 1024
```

Note that this uses the server key and server certificate generated in the example for the Server PFX field of the remote debug server configuration. The client certificate needs to be installed on the client machine.

On Windows, generate a client.pfx file:

```
openssl pkcs12 -export -out client.pfx -inkey client_key.pem -in
   client_cert.pem
```

Import client.pfx into the windows certificate store: double click on the pfx, add to Personal certificates for the current user.

On macOS, generate a pkcs12 file:

```
openssl pkcs12 -export -out client.p12 -inkey client_key.pem -in
   client_cert.pem
```

Double click on the file to add it to the keychain.

You can find more details about this in the CURL documentation at:

https://curl.haxx.se/libcurl/c/CURLOPT_SSLCERT.html

Note the Client Certificate parameter is the value passed to the CURL option CURLOPT_SSLCERT.

On Windows, the client certificate parameter is a path expression to a certificate store e.g.

```
CurrentUser\MY\afe2179599460d20da08c12e8c328d84bd300735
```

where afe2179599460d20da08c12e8c328d84bd300735 is the thumbprint viewed by double clicking on certificate in the MMC (MMC certificate snap-in view, details tab, thumbprint field).

On macOS, you can specify either the path of the p12 file, or the keychain name of the client certificate.

User Name

If the server uses HTTP basic authentication, the user name required for that.

Password

If the server uses HTTP basic authentication, the password required for that. Alternatively, you can leave this empty, and the client will prompt for the password when it is required.

Server Connection Logging

You can monitor the client connection to the Remote debugging server, which allows you to highlight any connection problems. You can enable logging in the remote debug client window (or in the config file in the logConnectionSetup item).

If enabled, the Remote Debug Client writes a log file named <session name>.htm to the logs/remotedebug folder, containing a log of what occurred when attempting to connect to the remote debug server. Note that the log is not written until the connection closes.

Preparing Code For Remote Debugging

You have to enable remote debugging in the library, and in the task instance (remote or standard for fat client), by setting the $remotedebug property.

Library

By default, a library cannot be debugged by the remote debug client, meaning that when the remote debug client connects to a server, the library will not appear in the client interface. If you wish code in a library to be debugged with the remote debugger, you need to set a new library property, $clib.$remotedebug: if true, remote debugging of this library is allowed, but it cannot be set to true in an always private library, which means you must set this property to true before making the library always private.

Task

Setting $clib.$remotedebug allows the library and its classes to appear in the remote debug client interface. This allows you to browse the code and set breakpoints. However, only tasks and remote tasks marked for remote debugging will react to these breakpoints. This provides more control over debugging, and specifically in the multi-threaded server, it prevents one breakpoint from stopping every client that hits it.

To mark a task or remote task for remote debugging, set the $remotedebug property of the task instance to kTrue.

In addition, you can set this property of a remote task by adding a query string parameter to the URL used to open a JavaScript client form or execute an ultra-thin request: omnisRemoteDebug=1, for example:

http://127.0.0.1:5981/jschtml/jsDragDrop.htm?omnisRemoteDebug=1

Remote Debugger Interface

Opening a Session

To use the remote debugger client after configuring a session, click on the Remote Debug Client node in the Studio Browser tree, click on the Open Session hyperlink, and then click on the hyperlink for the session you want to use.

This will cause the client to establish a WebSocket connection to the server. While the connection is being established, progress is displayed in the browser panel, although this is usually very quick. In addition, a Cancel Open Session hyperlink is displayed while the connection is being established.

Browsing Libraries

After the session opens, libraries marked for remote debugging appear in both the browser tree and the browser panel.

The Remote Debug Client child nodes have similar behavior to the Libraries node child nodes, so they include both libraries and folders within the libraries. When you select a child node, the browser panel updates to show the content of that node - this comprises a list of all classes that can contain code.

While any node in the remote debug client sub-tree is selected, a Close Session hyperlink is displayed. In addition, if a single class is selected in the browser panel, a hyperlink named "Open debug window" is displayed. You can click on this (or double click on a class in the panel) to open the remote debug window for the class.

Finally, if the server is paused, waiting to run startup, the hyperlinks include a link named "Run Startup" that can be used to tell the server to carry on and run its startup processing.

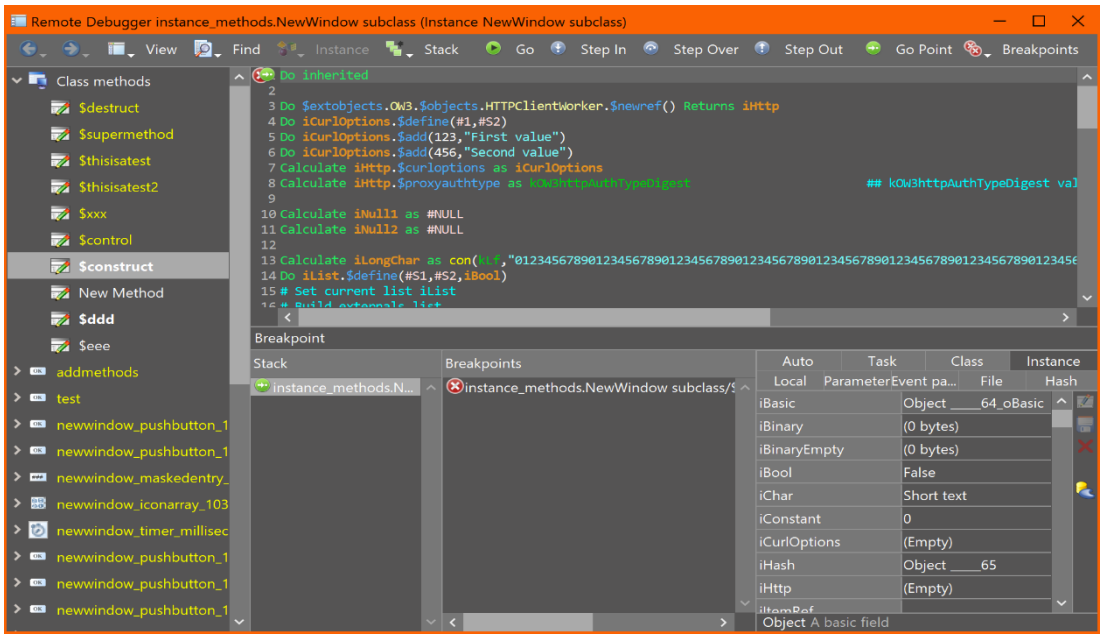
Save Window Setup for the browser window remembers column positions for the list view of the remote debug client panel.

The status bar of the browser window includes the name of the currently open session.

The Remote Debug Window

The remote debug window has a similar layout to the method editor. The main difference is that it always shows the debug panel (there is no editor panel), and in the bottom right-hand corner there is a new variable panel rather than the watch panel.

The window shares both its fonts and keyboard shortcuts with the method editor. So if you open the Fonts… dialog from either of these windows, you are editing the same configuration information (stored in keys.json under methodEditorAndRemoteDebugger).



Remote Debugger Toolbar

From left to right, the toolbar controls are as follows. Note that there is no configuration mechanism to change these.

Back

Navigates to the previous method in the history stack for the window. Note that unlike the method editor, there is a separate history stack for each remote debug window. This allows operations such as open superclass methods and open specified class to operate within the context of a single window, and also works more appropriately when you have several windows all paused at a breakpoint.

Forward

Navigates to the next method in the history stack for the window.

View

Open specified class is similar to the Modify Specified Class command in the method editor - when a method line is selected, it opens the method referred to by the command, if one can be identified.

In addition to a context menu command in the tree, the View menu also has a command to go to the superclass methods if relevant (also available on the Modify menu for the method editor).

Find

This allows you to perform find operations on the currently selected method.

Instance

The remote debug window can be associated with an instance. This allows you for example to view instance specific methods or objects that have been added during runtime execution. The instance menu allows you to close the instance, detach the debugger from the instance, or attach the debugger to an instance. Note that this menu is disabled as soon as the remote debug window becomes associated with some executing code (by hitting a breakpoint).

Stack

When execution is paused this allows you to select items on the call stack, or clear the stack.

Go, Step In, Step Over, Step Out

Like the standard method editor, these commands allow you to start execution (Go) and step through your debuggable code. The step commands step until the next

debuggable command, so if code which is not debuggable is encountered execution will not pause there.

Go Point

This allow you to set the go point to a different line in the method at the top of the call stack.

Breakpoints

This allows you to manage breakpoints. Note that the method editor has also been changed to remove individual Breakpoint and One-time breakpoint buttons, and use a similar menu to this for consistency.

The Set Condition… command allows you to set a condition on a breakpoint. The condition is a calculation that must evaluate to true for the breakpoint to pause execution. The condition dialog provides some code assistance, by using variable names (of task, class, instance, local, parameter and event parameter variables) present in the currently displayed method.

Note that the code panel and the breakpoint panel both provide alternative ways to work with breakpoints, in a similar way to the method editor - so the left column of the code panel can be used to set and clear breakpoints, and the breakpoints panel has a context menu to do this. Set Condition… is not available in the breakpoints panel context menu, because the method affected may not currently be displayed.

Variable Panel

The variable panel in the Method Editor will be populated while debugging your code remotely, and allows you to view and modify variables. (The Variable panel was introduced for Remote Debugging in Studio 10.0, but is now available in the standard Method Editor in Studio 10.1: it is described in detail in the Studio 10.1 section in this manual).

Keeping the Client in Step with the Server

You should bear in mind that the set of libraries and instances being debugged can change on the server. Omnis keeps the client up to date with the server using a combination of notifications sent from the server, and lazily applied updates to the client. For example, if a library closes on the server, the client is informed, and it updates the interface to reflect this - this means it removes it from the browser tree, and closes any remote debug windows for classes in the library. However, if a method changes on the server, the client will not receive the updated method until it requests it again - note that each time the client performs a debug operation, e.g. step over, the client will request the method when the action completes - if the method has changed on the server, the client will receive a new copy as part of the step action.

Execution

Execution Contexts

An execution context is either the main thread or a remote task instance. When execution suspends for an execution context, the remote debug client looks for **the** debug window associated with the context, and uses that. If there is no debug window for that context, the client looks for a suitable open remote debug window for the class, and if one exists that is not associated with a context it will use it, and associate the window with the context; otherwise the client opens a new window and associates it with the context. Once a window is associated with an execution context, all debugging for that context occurs in that window. A window associated with an execution context can only be closed if execution is not currently paused.

This approach means you can be simultaneously debugging several remote task instances for example. Each execution context uses a single window.

All in one process

As stated earlier, the remote debug client and server can be the same process. In this case:

When execution suspends in the main thread, the remote debug window for the main thread context becomes fully modal.

You cannot debug code running in a critical block in the multi-threaded server.

Errors

If an error occurs during Omnis code execution, e.g. Open window instance with a bad window name, and the line of code causing the error is remotely debuggable, the remote debugger pauses execution at the line causing the error.

Local Debugger

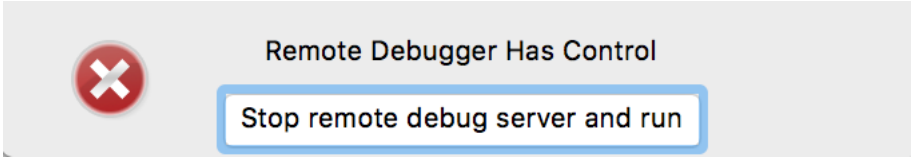
While the remote debugger is attached to a copy of Omnis, the local debugger is disabled in that copy. This also affects the ability to right click and view variable values.

Omnis Language

There is a new sys() function, sys(238) that returns a Boolean which is true if the remote debug server has been started.

Remote Debugger In Control

When the remote debug server and client are not the same process, and execution is suspended for the main thread on the server, the following window appears on the server (this does not apply to the headless server):



While this window is displayed, the only action that can be performed on the server is a click on the button to stop the remote debug server, and run (meaning execution continues from where it was paused).

Note that if you do choose to stop the server, then the session will close on the client, and all remote debug windows open on the client will close. If you subsequently restart the server (without restarting Omnis), and open a new session from the client, any breakpoints set for the previous session will still be set.

Remote Objects

The **Remote Object** is a new library class type. Remote Object classes are Object classes that are instantiated and executed entirely on the client, in the JavaScript Client. Each Remote Object class instance has a JavaScript object “class” that directly corresponds to it on the client.

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

Creating Remote Objects

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

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

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

The inheritance tree includes a node for remote object classes.

Omnis Language

Library Notation

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

Variables

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

Creating Instances

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

- a local variable of type object in a client-executed method
- or a remote form instance variable of type object.

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

Behavior

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

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

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

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

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

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

$cwind for remote objects

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

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

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

Code Generation

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

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

Web and Email Worker Objects

JavaScript Worker Object

The node.js framework contains many open source third-party modules that can be used from inside your Omnis code: node.js is now embedded into Omnis Studio. The new **JavaScript Worker Object** allows you to execute JavaScript methods inside node.js by making the request in Omnis code by calling a worker object method, and receiving the results via a worker callback. For example, the library 'xml2js' is included in Omnis Studio, which converts XML to JSON: in addition, support for ZIP can be added using the node.js jszip module, which is described at the end of this section.

Enabling Javascript Methods

There is a new JS file, ow3javascript.js, located in the clientserver/server/remotedebug program folder, that is the entry point for all method calls: on macOS, the remotedebug folder is in the Resources folder in the Omnis.app bundle. Method calls arrive as an HTTP request from Omnis, and respond with their results as HTTP content. A worker executes methods sequentially.

Omnis has a simple structure where you can write a JavaScript module containing one or more methods, and then call methods via their module and method name.

In the Omnis data folder, there is a new folder called node_modules, where modules required by ow3javascript.js are located. You can install additional node.js modules in this folder using the npm -i command when running in the folder - these might be modules for which you want to provide an interface from Omnis.

There are two key files in this folder, which must always be present:

omnis_calls.js - a module which provides an interface for methods to return their results to Omnis.

omnis_modules.js - a module which provides a table of modules that can be called from Omnis.

There are also two example module files, omnis_test.js and omnis_xml2js.js. These provide Omnis modules named test and xml2js. Each module file must have an entry in omnis_modules.js. Each module file provides a table of methods that can be called from Omnis.

Note that node_modules in the data folder may not be considered suitable for deployment, since the data folder is writeable. The worker provides the ability for you to structure things differently when you deploy your application, but is fine for development.

Creating the worker

The sub-type of the external object is OW3 Worker Objects\JAVASCRIPTWorker. You can use either an Object variable or an Object Reference variable, either directly if you set $callbackinst to receive results, or by subclassing the external object with an Omnis object.

Properties

The JavaScript worker only has the standard worker properties: $state, $threadcount, $errorcode and $errortext.

Methods

Called Methods

$init()

$init([cPath, bDebugNodeJs=kFalse])

Initialize the object so it is ready to execute JavaScript method calls. Returns true if successful. You must call $init() before any other methods.

- **cPath**
  allows you to override the default NODE_PATH module search path set by the worker. The default path is <Omnis data folder>/node_modules. If you override this path, the various JavaScript modules that are mandatory for the worker to operate must still be able to be located.
- **bDebugNodeJs**
  is a Boolean that indicates if you want to be able to debug node.js, for example using Chrome. It is possible that you may not be able to start the worker if you set this for more than one active JavaScript worker, as node.js requires a debug port to be available. To debug the JavaScript in Chrome, navigate to chrome://inspect, and then open the dedicated debug tools for node.js via the link.

$start()

$start()

Runs the JavaScript worker in a background thread. Returns true if the worker was successfully started.

After you call $start(), the background thread starts up a node.js process which will process JavaScript method calls. You can make multiple method calls to the same process, so there is no need to call $start() frequently, which means the overhead of starting the node.js process is minimal.

$cancel()

$cancel()

Use this to terminate the node.js process. Any in-progress method calls may not complete.

$callmethod()

$callmethod(cModule, cMethod, vListOrRow [,bWait=kFalse, &cErrorText])

Call the method passing it a single parameter which is the JavaScript object representation of vListOrRow. Optionally wait for the method to complete. Returns true if successful.

cModule and cMethod identify a module, and a method within the module, to call, as described above.

vListOrRow will be converted to JSON, and passed to the method as its parameter. This means that you should be aware of data that will not map to JSON, and avoid trying to pass that to $callmethod.

bWait indicates if the caller wishes to suspend execution until the method completes. If you use bWait, then a completion callback will occur before $callmethod returns.

cErrorText receives text describing the error if $callmethod fails.

Callback Methods

$cancelled

Override this to receive a notification that the request to cancel the worker has succeeded.

$workererror

$workererror(wError)

Override this method to receive reports of errors from the worker that are not related to calling a method, e.g. failure to start node.js. The worker thread exits after generating this notification.

wError has two columns, an Integer named errorCode and a Character string named errorInfo.

$methoderror

$methoderror(wError)

Override this method to receive reports of the failure of an attempt to call a method with $callmethod.

wError has two columns, an Integer named errorCode and a Character string named errorInfo.

$methodreturn

$methodreturn(wReturn)

Method called with the results of a call to $callmethod.

wReturn is a row. If the JavaScript method returns an object, this is the Omnis equivalent of the object, created by converting the JSON to a row. If the JavaScript method returns some other data, e.g. a picture, this is a row with a single column named content, which contains the data returned by the method.

Example: Adding ZIP support

You could add support for ZIP files. To do this, install the node.js jszip module by running the npm command in the node_modules folder:

```
npm i jszip
    (the npm command is installed with node.js, available on the web)
```

Edit omnis_modules.js as by adding this line after the other require() lines:

```
const zipModule = require('omnis_zip.js');
```

Add this entry to the moduleMapClass:

```
zip(method, obj, response) {
   return zipModule.call(method, obj, response);
}
```

You can then make calls from Omnis code as follows:

```
Do lRow.$cols.$add("path",kCharacter,kSimplechar)
Calculate lRow.path as iZipPath
Do iJS.$callmethod("zip","loadZip",lRow,kTrue,lErrorText) Returns lOK
```

POP3 Worker Object

A POP3 Worker Object has been added to the OW3 Worker Objects. The POP3 worker is similar to other OW3 workers, in that you pass an action to $init and action specific parameters, and use $run or $start to execute the request. There is a new sample app in the HUB in the Studio Browser.

The $init method has the following syntax:

- **$init()**
  $init(cURI,cUser,cPassword,iAction[,iMessageNumber,cPostCommand])
  Initialises the object so it is ready to perform the specified action using POP3. Returns true if successful

The $init parameters are:

- **cURI** The URI of the server,optionally including the URI scheme (pop3 or pop3s) e.g. pop3://pop3.myserver.com. If you omit the URI scheme e.g. pop3.myserver.com,the URI scheme defaults to pop3
- **cUser** The user name to be used to log on to the POP3 server
- **cPassword** The password to be used to log on to the POP3 server
- **iAction** A kOW3pop3Action... constant that specifies the action to perform
- **iMessageNumber** The message number to get (applies to actions kOW3pop3ActionGetMessage,kOW3pop3ActionGetHeaders and kOW3pop3ActionDeleteMessage)
- **cPostCommand** Only applies to action kOW3pop3ActionGetMessage.If not empty,a command to send to the server after getting the message.This would typically be RSET to undelete messages or QUIT to delete messages

The Actions are:

- **kOW3pop3ActionStat** Gets maildrop status. For a successful request, wResults has 2 columns returning the stat information, messageCount and maildropSize
- **kOW3pop3ActionList** Lists messages. For a successful request, wResults has a column named messageList which contains a 2 column list, with columns messageNumber and messageSize
- **kOW3pop3ActionGetMessage** Gets specified message. For a successful request, wResults has either a column rawData or columns headerList and mimeList (depending on the value of $splitfetchedmessage
- **kOW3pop3ActionGetHeaders** Gets headers for a specified message. For a successful request, wResults has either a column rawData or a column headerList (depending on the value of $splitfetchedmessage)
- **kOW3pop3ActionDeleteMessage** Deletes the specified message from the maildrop
- **kOW3pop3ActionQuit** Sends a QUIT command to the server to ensure that any messages marked for deletion are deleted

The worker has the following properties in addition to those supported by all OW3 worker objects:

Property	Description
$splitfetchedmessage	If true, worker splits fetched message into headers and MIME list for any content. Defaults to true. If false, the worker simply returns the raw fetched message data (Applies to kOW3imapActionFetchMessage and kOW3pop3ActionGetMessage)
$defaultcharset	Used by kOW3imapActionFetchMessage and kOW3pop3ActionGetMessage when there is no charset for a MIME text body part. The charset used to convert

	to character. Default kUniTypeUTF8.A kUniType... constant (not Character/Auto/Binary)
$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.
$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 STARTTLS or STLS). If it cannot be upgraded then the request fails. Defaults to false

CRYPTO Worker Object

A CRYPTO Worker Object has been added to the OW3 Worker Objects to allow you to perform encryption and decryption of data. The encryption types you can use include AES, Camellia, DES, and Blowfish.

The CRYPTO worker is similar to other OW3 workers, in that you pass an action to $init and action specific parameters, and use $run or $start to execute the request. There is a new sample app in the HUB in the Studio Browser to demo the CRYPTO Worker Object.

To use the worker object, create a variable with its subtype set to the CRYPTOWorker object which is contained in the OW3 Worker Objects group in the Select Object dialog, or subclass the external object.

The CRYPTO worker has the following methods:

- **$start, $run and $cancel**
  Standard worker methods
- **$completed** and **$cancelled**
  Standard worker completion methods
- **$makerandom**
  $makerandom(iBytes,&xRandomData) generates some random data with the specified length in bytes. This is suitable for use as an encryption key or initialization vector. Returns true if successful (if an error occurs, the standard properties $errorcode and $errortext describe the error).
  iBytes is the number of bytes of random data to generate
  xRandomData is a binary variable that receives the random data

As with the other worker objects you can use the $init method with various input parameters to initialize the object, while the action can be to Encrypt or Decrypt:

- **$init**
  $init(iAction, iEncryptionType, iCipherMode, iPadding, xKey, xIV, vInputData [,cOutputPath]) initialises the object ready to perform the encryption or decryption. Returns true if successful

iAction can be either:

- kOW3cryptoActionEncrypt
  Encrypt the data using the specified encryption scheme and parameters
- kOW3cryptoActionDecrypt
  Decrypt the data using the specified encryption scheme and parameters

iEncryptionType can be one of:

- kOW3cryptoTypeAES
  AES encryption. Key size must be 128, 192 or 256 bits

- kOW3cryptoTypeCamellia
  Camellia encryption. Key size must be 128, 192 or 256 bits
- kOW3cryptoTypeDES
  DES encryption. Key size must be 64 or 192 bits. Uses Triple DES if key size is 192 bits
- kOW3cryptoTypeBlowfish
  Blowfish encryption. Key size must be 128 bits

iCipherMode can be one of:

- kOW3cryptoCipherModeCBC
  CBC (Cipher Block Chaining)
- kOW3cryptoCipherModeECB
  EBC (Electronic Code Book)
- kOW3cryptoCipherModeCTR
  CTR (Counter). Not supported for kOW3cryptoTypeDES

iPadding can be one of:

- kOW3cryptoPaddingNone
  No padding (use this for cipher modes other than CBC and EBC)
- kOW3cryptoPaddingPKCS7
  PKCS7 padding
- kOW3cryptoPaddingOneAndZeros
  One and zeros padding (ISO/IEC 7816-4)
- kOW3cryptoPaddingZerosAndLen
  Pad with N-1 zero bytes followed by a byte with value N, where N is the number of padding bytes (ANSI X.923)
- kOW3cryptoPaddingZeros
  Pad with N zero bytes, where N is the number of padding bytes

Note that PKCS7 is the most common and allows binary data to be correctly decrypted, for example, kOW3cryptoPaddingZeros can lead to extra bytes being stripped from decrypted binary data.

xKey is a binary containing the key. See the encryption types for details of supported key lengths.

xIV is a binary containing the Initialization Vector (IV) (random data that can be used to make the same encrypted data different when using the same key). This must be 8 bytes long for DES and Blowfish, and 16 bytes long for AES and Camellia.

vInputData is the data to be encrypted. Either a character value which is the pathname of the file containing the data to encrypt or decrypt, or a binary variable containing the data to encrypt or decrypt.

cOutputPath is:

- Either omitted or empty meaning that the encrypted or decrypted data is supplied to $completed in the data column of the results row parameter (this column has type binary)
- Or the pathname of a file (which must not already exist) to which the worker will write the encrypted or decrypted data

The results row passed to $completed has 3 columns: errorCode, errorInfo and data. The error columns behave in the same way as the other OW3 workers: note that a negative error code is a code returned by the mbedTLS library. The data column is only used when cOutputPath was omitted when calling $init.

The worker has properties as follows:

- $errorcode, $errortext, $state and $threadcount
  Standard worker properties

- $useexplicitiv
  If true, a random IV is added to the start of the data when encrypting or the first IV size bytes is removed from decrypted data. Only applies to CBC cipher mode. This means that a user can decrypt data without knowing the IV (any IV can be used for decryption, which will result in just the first IV size bytes being decrypted incorrectly, because of the way the CBC cipher mode works)

HASH Worker Object

A HASH Worker Object has been added to the OW3 Worker Objects to allow you to hash data using SHA1, SHA2, SHA3, and RIPEMD hash types, which are primarily for signature purposes, while PBKDF2 is available for password hashing. You use the $inithash method to initialise the object, passing the binary or character data to be hashed, and the hash type represented by a constant, as follows:

Hash type	**Constant**
SHA1	kOW3hashSHA1
SHA2 incl hash output 256, 384, 512	kOW3hashSHA2_256
	kOW3hashSHA2_384
	kOW3hashSHA2_512
SHA3 incl hash output 256, 384, 512	kOW3hashSHA3_256
	kOW3hashSHA3_384
	kOW3hashSHA3_512
RIPEMD_160	kOW3hashRIPEMD_160
PBKDF2	kOW3hashPBKDF2

You can use $initverifyhash to verify or compare some data against previously hashed data generated using $inithash. Having called the $init… methods, you can use $run or $start to execute the request.

To use the worker object, create a variable with its subtype set to the HASHWorker object which is contained in the OW3 Worker Objects group in the Select Object dialog, or subclass the external object. There is a new sample app in the HUB in the Studio Browser to demo the HASH Worker Object.

The HASH worker object has methods as follows:

- **$start, $run** and **$cancel**
  Standard worker methods
- **$completed** and **$cancelled**
  Standard worker completion methods
- **$inithash()**
  $inithash(vData,iHashType,wHashParameters) initialises the object so it is ready to hash data using the specified hash type
  vData is the data to hash. Either binary or character. A binary value is used directly. Otherwise, for PBKDF2 the worker converts character data to UTF-8 before generating the hash, and for other hash types, a character parameter is the pathname of the file containing the data to hash.
  iHashType the hash type, a kOW3hash… constant.
  wHashParameters A row of parameters that control the hash. For all except PBKDF2, an empty row(). For PBKDF2, a row with 3 integer columns in the order saltLength, keyLength, iterations.
  Salt length - the length of the random salt (generated by the worker). A good value for this is 16. Must be 8 to 64 inclusive

Key length - the length of the hashed key to be generated. A good value is 32. Must be between 16 and 256 inclusive
Iterations - the number of iterations to perform to generate the hash. A good value for iterations is at least 100000 - the higher the value, the more secure the hash, traded off against a longer execution time. Must be between 1 and 256000 inclusive

- **$initverifyhash()**
  $initverifyhash(vData,iHashType,vHash) Initialises the object so it is ready to generate the hash for vData using iHashType and compare it against previously generated vHash (vHash includes the hash parameters used to generate the original hash), e.g. you could create a hash using $inithash and store that for future use. To verify a password or document you call $initverifyhash, with vData as the password or document to verify, and vHash as the stored hash from your call to $inithash.

After calling one of the $init… methods, you call $run or $start. The results row passed to $completed has 3 columns: errorCode, errorInfo and data. The error columns behave in the same way as the other OW3 workers - note that a negative error code is a code returned by the mbedTLS library. The data column is only used for $inithash() and it contains the generated binary hash, provided that no error occurred – you may want to convert this to base64 before storing it, but note that if you do this you will need to convert it back to binary before using it to verify data. For $initverifyhash(), the errorCode is zero if and only if the hash was successfully verified.

The worker has properties as follows:

- $errorcode, $errortext, $state and $threadcount
  Standard worker properties

FTP Worker Object

Support for SFTP (Secure FTP) has been added to the OW3 FTP Worker Object. There are some differences in functionality when using SFTP:

1. You use URLs of the form SFTP:// to request an SFTP connection.
2. You must explicitly select a server character set - kUniTypeAuto will cause the worker to return an error.
3. The append file action is not supported.
4. If you have written code that uses the execute action, be aware that SFTP servers support different commands to FTP servers.
5. The remaining actions work as expected.

In addition, there are some new properties and methods, primarily related to how a connection is authenticated. SFTP does not use TLS, so the secure options related to that only affect FTPS and FTP connections to be upgraded to TLS.

The FTP worker object has some new properties:

- **$sshenablecompression**
  If true, SSH compression is enabled for SFTP connections, resulting in a request to the server to enable compression; the server may ignore the request. Defaults to false
- **$sshknownhostsfile**
  The full pathname of the SSH known hosts file used for SFTP. Defaults to the path of clientserver/client/ow3_sftp_known_hosts in the Studio tree. Set this to empty to allow connections (insecurely) to any host
- **$sshknownhostsaction**
  A sum of kOW3sshKHAction... constants (default kOW3sshKHActionReject) specifying what occurs if $sshknownhostsfile is present, and a connection is to be made to a server not in the file, or a server in the file with a host key mismatch

- **$sshauthtypes**
  A sum of kOW3sshAuthType... constants (default kOW3sshAuthTypePublicKey+kOW3sshAuthTypePassword+kOW3sshAuthTypeHost+kOW3sshAuthTypeAgent) specifying the allowed authentication types when establishing a connection to the server

The FTP worker object has some new methods:

- **$getsshoptions()**
  $getsshoptions([&cServerPublicKeyMD5,&cClientPublicKeyFile,&cPrivKeyFile,&cPrivKeyPassword]) gets the options that affect how SSH connections are established for SFTP
- **$setsshoptions()**
  $setsshoptions([cServerPublicKeyMD5='',cClientPublicKeyFile='',cPrivKeyFile='',cPrivKeyPassword='']) sets the options that affect how SSH connections are established for SFTP. The parameters are:
  cServerPublicKeyMD5:The 128 bit MD5 checksum of the server's public key (supplied as a 32 character ASCII hex string).If not empty,the SFTP client will reject the connection to the server unless the MD5 checksums match
  cClientPublicKeyFile:The pathname of the client's public key. If empty,the client will try to compute the public key from the private key
  cPrivKeyFile:The pathname of the client's private key file
  cPrivKeyPassword:The private key file password

Some or all of the SSH options may be optional, depending on the authentication type chosen.

JSON Components

JSON Component Editor

The JSON Component editor has been enhanced.

- The Build option now adds update markers and gives user opportunity to update JavaScript if the markers already exist.
- The JSON file for a component must be named control.json. The editor will prompt/warn you if the JSON control file is not named control.json.
- Double quote and backslash characters are now escaped when saving all desc and property initial value items.
- When setting the initial value for boolean type properties, values of ‘true’ or ‘kTrue’ are overridden with 1. In addition, there is extra validation for min, max and initial values.
- When setting extconstant and intconstant members for properties, only one can be selected at a time (both cannot be selected). If intconstant is selected, a constant name such as kPlain entered into constrangestart or constrangeend is converted to its ident value.
- The editor now prompts you to save if changes have been made on Build and Reload, as well as when closing the editor.

Report Programming

Report Working Messages

The Omnis preference $disablereportworkingmessage has been added to allow you to disable working messages for reports, which you might want to do when printing to a Print Review window.

- **$disablereportworkingmessage**
  If true, the 'Sending report to...' working message is not shown when printing a report.

This property only applies to reports being printed on the main thread (as reports in JavaScript client threads do not show a working message). Note that you cannot cancel the report if you set this property to true. You may also need to use $modes.$fixedcursor and $modes.$ccursor if you want to display a cursor other than the busy cursor while printing the report.

Copy from Print Preview

You can now use the tab key to tab through the page list, page and search field in the Print Preview screen. Therefore, to copy content from the preview screen, you can tab to the page if necessary, press Cmd+A to Select all and then Cmd+C to copy content.

When the page has the focus, you can use the Escape key to clear the selection.

PDF Destination

The PDF report destination now renders hairlines correctly due to a fix in this version. The alternative of using the Printer report destination with $macosdesttype set to PDF uses bitmaps to render background objects, and their appearance in a scaled PDF will vary depending on the scaling factor and the algorithm used by the PDF viewer to scale the bitmap.

Libraries

Export Libraries to JSON

You can now export multiple libraries to JSON via the Studio Browser. To do this you need to select the Libraries node on the Studio Browser and select one or more libraries in the Library pane. The 'Export to JSON' option will appear allowing you to export the selected library/libraries to the JSON export folder.

Save Window Setup

The Save Window Setup option is now available in the context menu option for JSON Export/Import and the Errors/Warnings window.

Default Library Internal Name

Prior to this version, file paths on the Windows platform were converted to upper case when Omnis opened a file, such as a library file. This resulted in the default internal name for a library being upper case on the Windows platform.

In this version, file names and paths are no longer converted to upper case when opened on Windows, so the default internal library name will have the case of the library name. This is consistent with Omnis running on the macOS and Linux platforms.

Color Themes and Appearance

Appearance Subgroups

The colors listed in the appearance.json file, and the associated theme files, have been grouped into subgroups, to make it easier to locate appearance settings related to specific areas or controls in Omnis, and include the following subgroups:

```
"checkRadio",
"compareTool",
"dropList",
"edit",
"generalButton",
"groupBox",
"header",
"IDEgeneral",
"IDEmethodEditor",
"IDEmethodSyntax",
"list",
"menu",
"pagePreview",
"pushButton",
"scrollbar",
"system",
"tabPane",
"toolbar",
"tooltip",
"tree"
```

The subgroups prefixed with IDE refer to colors in the IDE only, such as the Method Editor, so any changes you make to colors in these groups are only visible in the IDE, not the Runtime. All other theme subgroups affect colors in the IDE and the Runtime version of Omnis Studio. When you edit the $appearance preference in the Property Manager you will see the new subgroups, so to edit colors in the Method Editor syntax you can open the “IDEmethodSyntax” group.

All new appearance and theme files generated in Omnis Studio 10.0 will contain the new subgroups. Any theme files created in Studio 8.1.x (without the subgroups) will continue to work in the new version. You may prefer to use the new appearance and theme files with the subgroups for all new applications.

Searching Colors & Themes

A search field has been added to the dialogs for the $appearance preference and the new $keys preference (these dialogs are opened by clicking on the property/preference in the Property Manager). As you type into the search box, Omnis will highlight any matching lines in the Property Manager and scroll to the first match. Tabbing from the search field sets the focus in the grid to the first match.

Studio Browser

Class Browser

File class filter

The keyboard shortcut for the File class filter in the Studio Browser is now Ctrl+Shift+E. The old shortcut for file classes was Ctrl+Shift+F which now opens the Find and Replace window (located on the Edit menu).

Copy Class Name

You can copy the name of a selected class to clipboard by pressing Ctrl-N or via the Copy Name option on the Class Browser context menu: for multiple selected classes the names are copied in a list.

iSQL Tool & Query Builder

You can now resize the font in the Interactive SQL (results pane) and Query Builder windows using the Ctrl+/Ctrl- shortcut keys.

Superclass Methods

A new "Superclass methods..." command has been added to the Studio Browser context menu for a class to allow you to jump to the methods of the superclass on the class.

Find and Replace

Find Log

There are a number of enhancements in the Find log window. You can sort the Find log list by clicking on the header buttons: this is in addition to the current functionality (Studio 8.1.x) where you can sort the list by either of the first two columns by using the context menu. The context menu now also has an item to sort the last column. Keyboard searching of the Find log list now searches column two.

This enhancement means you can locate entries of a particular type more easily in the Find log, by clicking on the Type header to sort the list, and then typing the type name to search the list.

Localization

Localizing Built-in Strings

Some new strings have been added to the jOmnisStrings object in the JavaScript Client which allows you to localize (translate) the strings, if required. The strings prefixed "ctl_" (except ctl_tree_invmode) have been added in this version and relate to strings that appear on the File and Tree list controls. The following is a complete list of strings, including the new ones.

String object	String text (English default)
comms_error	An error has occurred when communicating with the server. Press OK to retry the request
comms_timeout	The server has not responded. Press OK to continue waiting
ctl_dgrd_id	You cannot use \x01 as a list column name for a list bound to a data grid
ctl_dgrd_other	(1 other)
ctl_dgrd_others	(\x01 others)
ctl_file_batchsizeerror	Total batch of files is larger than maximum allowed upload size (\x01)
ctl_file_batchsizetext	\x01 of \x01
ctl_file_downloaderror	Download error
ctl_file_filesizeerror	File size is larger than maximum allowed upload

String object	String text (English default)
	size (\x01)// \x01 //
ctl_file_filesizetext	\x01 of \x01
ctl_file_filesuploaded	\x01/\x01 files
ctl_file_stopbutton	Cancel upload
ctl_file_uploadbutton	Upload
ctl_file_uploaderror	Upload error
ctl_file_uploadmultipletitle	Upload files
ctl_file_uploadstopped	Upload stopped
ctl_file_uploadtitle	Upload file
ctl_subf_params	Control \x01: $parameters cannot be assigned at runtime
ctl_tree_badgnl	Control \x01: Internal error calling get node line for dynamic tree
ctl_tree_badident	Control \x01: You cannot use \x01 as a tree node ident - tree node idents must be a non-zero positive integer
ctl_tree_dupident	Control \x01: The tree already has a node with ident \x01 - tree node idents must be unique
ctl_tree_invmode	Control \x01: Invalid data mode for tree
disconnected	You have been disconnected. Refresh or restart application to reconnect
error	Error
local_storage_unavailable_error	Unable to access localStorage (perhaps cookies are disabled?).//The application will not run.
omn_cli_badobj	object $objs.\x01 does not exist
omn_cli_callprivate	callprivate cannot call \"\x01\"://Exception: \x01
omn_cli_cgcanassign	cannot use $canassign for row section object \"\x01\" in complex grid because it has exceptions
omn_form_addbadpage	Parent page number \x01 not valid for paged pane \"\x01\
omn_form_addbadparent	Parent object \"\x01\" for add control is not a paged pane
omn_form_addcg	Cannot add control to parent object \x01 contained in complex grid
omn_form_addparent	Cannot find parent object \x01 for add control
omn_form_addsrc	Cannot find source object \x01 for add control
omn_form_ctrlinst	Failed to install the control \x01. Possible missing class script
omn_form_nofile	There is no file with the specified ident (\x01)
omn_form_noinstvar	Instance variable does not exist (\x01)

String object	String text (English default)
omn_form_readfileerror	Error \x01 occurred when reading the file with ident \x01
omn_inst_assignpdf	Assign PDF: HTML control \"\x01\" not found
omn_inst_badformlist	\x01: Invalid formlist
omn_inst_badparent	\x01: Invalid parent for subform set
omn_inst_badpn	\x01: Paged pane does not have page \x01
omn_inst_badpp	\x01: Cannot find the paged pane with name \"\x01\
omn_inst_badservmethcall	Cannot make server method call when waiting for a response from the server
omn_inst_badsfsname	\x01: Invalid or empty name for subform set
omn_inst_cliexcep	Exception occurred when executing client method://
omn_inst_dupsfsname	\x01: A subform set with this name already exists
omn_inst_dupuid	Subform set already contains unique id \x01
omn_inst_excep	Exception occurred when processing server response://
omn_inst_excepfile	File \"\x01\" Line \x01//
omn_inst_formloaderr	Failed to load form data for \x01. Server returned \x01
omn_inst_formnum	Invalid form number. Parameter error \x01
omn_inst_objnum	Invalid object number. Parameter error \x01
omn_inst_respbad	Unknown response received from server
omn_inst_senderr	Failed to send message to server
omn_inst_sfsnotthere	\x01: A subform set with name \"\x01\" does not exist
omn_inst_xmlhttp	Failed to initialize XMLHttpRequest
omn_list_badaddcols	The argument count for $addcols must be a multiple of 4
omn_list_badrow	Invalid list row
omnis_badhtmlesc	Invalid HTML escape
omnis_badstyleesc	Invalid style escape sequence
omnis_convbad	Error setting \x01: variable type \x01 not supported by JavaScript client
omnis_convbool	Error setting \x01: data cannot be converted to Boolean
omnis_convchar	Error setting \x01: data cannot be converted to Character
omnis_convdate	Error setting \x01: data cannot be converted to Date
omnis_convint	Error setting \x01: data cannot be converted to Integer

String object	String text (English default)
omnis_convlist	Error setting \x01: data cannot be converted to List
omnis_convnum	Error setting \x01: data cannot be converted to Number
omnis_convrow	Error setting \x01: data cannot be converted to Row
omnis_escnotsupp	Text escape not supported by JavaScript client
switch_off	OFF
switch_on	ON

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.

Deploying your Web & Mobile Apps

Updating the SCAF

In previous versions you had to quit Omnis and delete the SCAF files in order to force Omnis to rebuild the SCAF. Now you can do this without having to quit Omnis, by clicking on the ‘Omnis Studio’ node in the Studio Browser and selecting the ‘Update Omnis SCAF’ option.

Headless Omnis Server OSAdmin

There is a new member "headlessDatabaseLocation" in the “server” section of the Omnis Configuration file (config.json) that allows you to specify the location of the database for the Headless Server admin tool. When populated, the admin tool will look for / create the SQLite database file in this location, rather than the default.

Server Logging

Folder location

The "folder" item in the “logToFile” section of the config.json file, which specifies the location of logs for the Omnis App Server, can now be a full path name (which must not end in a path separator character), and the end folder name will be created if it does not already exist. In previous versions, you could only specify a folder relative to the Omnis folder, but now a full path can be used which can be outside the main Omnis folder. For example:

```
"folder": "/Users/bd/Sites/logs"
```

would send the log to the specified folder, while

```
"folder": "logs"
```

would still be read as relative to the main Omnis folder (note no starting or ending path separator).

You must use / as the path separator on macOS and Linux, whereas, you can use / or \ on Windows.

Log count

The maximum for the "rollingcount" item in the "logToFile" section of the config.json file has been increased to 1024. The logtofile component uses a new log file every hour (or daily from 10.1), so the new max value would allow logs to be stored for up to six weeks, at which point the oldest logs would be deleted.

If there is an error initialising logging, the logtofile component also writes it to standard output when running on Linux.

Omnis Configuration

Template config.json

The template config.json located in the 'templates' folder in the 'Studio' folder has been renamed and is now called 'configtemplate.json'. The template file contains all possible configuration settings in Omnis: you can copy sections out of this file and add them to your copy of the config.json file in the Studio folder to configure specific parts of Omnis.

Editing config.json

You must close Omnis Studio before editing the Omnis Configuration File. If you do not close Omnis, then any changes you make will be lost.

SQL Programming

SQL Data Type Mapping

The data type mapping for Character columns between SQL table columns and Omnis Schema class columns has been improved. The definition of Character columns in Omnis schema classes has changed and now allow lengths from 0xffff to (100000000 - 1) to be stored correctly. In previous versions, the column sublen of 65535 or greater would have been mapped to 100000000.

This change also means that file classes now support the same lengths.

Omnis Programming

Object Variables

There has been a small but significant change in the way errors are handled when you open a library that tries to construct an Object variable and the Object class it is based on does not exist or is in a library that is not open. In this scenario, in this release you will get a runtime error and execution blocks with the following error:

```
E100101: Class not found when constructing an object variable
The class name is TESTB.oTestB
Either the class does not exist or it has the wrong type e.g. remote object
```

In previous versions in this case, you would have received the error "Class not found", but code execution would have continued which may have led to further errors in the application.

Private Methods

Private methods are excluded from the Notation Inspector when showing the methods of a protected class.

Web Services

ORA Properties and Methods

The $httpresponsecode property no longer returns the informational status codes 100-199.

Window Classes & Components

There is a new Round Button window class control and some enhancements to a number of other window class controls. In addition, support for drag and drop for external files in desktop apps has been extended.

Round Button

The **Round Button** is a new window class control that provides a graphical & highly configurable button for your window class forms: it is called RoundButt Control and is under the External Components tab in the Component Store. You can use the Round Button to show the progress of a process, or to show individual values in a group of data points such as percentages. The Round button control uses transparency, so requires a minimum of Windows 8 or higher.

The Round button has a number of properties which can be set at runtime to indicate progress.

- **$centerimage**
  an optional image which will be clipped inside the circular progress bar, including the amount specified in $progressgap.
- **$progressalpha**
  the alpha value of the progress bar: 0-255 with zero being transparent
- **$progresscolor**
  the RGB color of the progress bar
- **$progressgap**
  the gap between the inside of the progress bar and the center image
- **$progressstartangle**
  the starting angle of the progress bar: 0-359 with zero at the top
- **$progressvalue**
  the current value of the progress bar as a percentage: 0-100 with 100 being progress complete, that is, the progress bar is a complete full circle
- **$progresswidth**
  the width of the progress bar in pixels

As well as these properties to configure the appearance of the control, the control responds to a standard click which you add event processing to.

Drag and Drop

For Win and macOS

When you drag a file from the system and drop it onto a field that can accept files (its dropmode is kAcceptFileData) then the file extension or file type is now added as the third column of the pDragValue list parameter which is passed. For example, this may be a file extension .txt for a text file, or a Uniform Type Identifier on MacOS such as com.apple.mail.email for an email.

macOS only

On macOS, support for dragging and dropping from sources other than the file system has been added. If a third-party application supports the drag pasteboard on macOS, typically Omnis should be able to receive the data placed on the pasteboard during a drag and drop operation. The format of this data will vary between applications, but a couple of examples are discussed here.

Apple Mail

When dropping a single Apple mail onto an Omnis field which accepts system file information (dropmode is kAcceptFiles) the drag value passed in pDragValue on an evCanDrop or evDrop event will be a list with a single row. The first column will be a message URL value and the second column will specify a type of com.apple.mail.email. The message URL is formatted in the standard internet message format which contains the message ID. This can be used to identify the message and, for example, you could use AppleScript to get the body of the message.

Dropping multiple Apple mails results in an empty message URL on the pasteboard and this is passed in the first column of the drag value as "message:" with a type in the second column of com.apple.mail.email. If information about multiple emails is required, the field must accept file data (kAcceptFileData) and process the data passed as discussed below.

When accepting file data (kAcceptFileData), dropping either a single or multiple mail message will result in a pDragValue which is a list with a single row. For a single email the values in the first column will be a message URL containing message information and for multiple emails it will be an empty message URL. Both will have a type in the third column of com.apple.mail.email. The data in the second column will be available when an evDrop event is sent and it will be formatted as an XML property list with information for each mail message. For an evCanDrag event this will be the length of the data which will become available. The property list will be an array of dictionaries with an entry for each email. The key/value pairs in the dictionary will provide the email account, the id of the email, the mailbox and the subject of the email.

The XML property list can be read into an oXML DOM Document object, e.g.

```
#lBin - Binary
#lPropertyList - Binary
#lErrorText - Character
#lXML - Object ( DOM Document )

On evDrop
..
Calculate lBin as pDragValue.1.2
Do uniconv(kUniTypeUTF32,lBin,kUniTypeUTF8,lPropertyList,kFalse,lErrorText)
Do lXML.$loadbinary(lPropertyList,lErrorText)
..
```

Note that the internal format of the dragged data will be UTF32 therefore this needs to be converted to UTF8 using the uniconv function before it is used in the $loadbinary call.

Once the property list is represented by a DOM document object then it is possible to extract the information for each message from the object.

Apple Calendar

A calendar event can be dragged and dropped onto an Omnis field from the Apple Calendar application. However, the calendar description does not become available until the evDrop event is sent. As part of the evCanDrop event the pDragValue will contain a type entry in its first row with an Apple UTI of com.apple.ical.ics. If only requesting file information (kAcceptFiles), the evDrop drag value will contain a path to a temporary ICS file in its first column. That file describes the calendar data in the standard iCalendar format. The second column will show the type as ICS. The data can then be subsequently loaded from the file on disk and read into an Omnis iCalendar document object.

When accepting file data (kAcceptFileData), the second column will contain the ICS formatted data for the event. This can be read into an Omnis iCalendar document object, for example:

```
#lBin - Binary
#lCalendarEvent - Character
#lErrorText - Character
#lXML - Object ( DOM Document )

On evDrop
..
Calculate lBin as pDragValue.1.2
Do
   uniconv(kUniTypeUTF8,lBin,kUniTypeCharacter,lCalendarEvent,kFalse,lErrorText
   )
Do lDoc.$initwithdata(lCalendarEvent) Returns #1
..
```

Note that the internal format of the dragged data will be UTF8 therefore this needs to be converted to UTF32 using the *uniconv()* function before it is used in the $initwithdata call.

Once the calendar event is represented by an Omnis iCalendar object it is possible to extract information about individual components within the event.

List Control

There is a new color setting "alternatelinecolorplatforms" in the appearance.json file to allow you to enable alternating colors for list lines. This option is an integer that indicates the platforms on which the odd and even list row colors are used for relevant lists with background theme kBGThemeControl. The values are: 0 for no platforms, 1 for macOS (the default), 2 for Windows, 3 for macOS and Windows.

Pushbutton

On macOS from Studio 8.0.x pushbuttons flash when they are clicked; this is the default behavior for macOS buttons. You can disable this behavior using a new option in the Omnis confirguration file: under the macOS section in config.json file, set **macOSbuttonNewTextDrawingStyle** to false.

Headed List

You can now detect which column the mouse is over in a headed list. The function mouseover(kMHorzCell) now returns the column number of the headed list if the mouse is over the control.

Text Object

The behavior of the $vertcentertext property for single- and multi-line Text objects has been modified. If true, single-line text (or any text in a kText background object) is vertically centered in the height of the field. If false, the text is vertically positioned according to the rules in previous versions of Omnis Studio.

Gif Control

The behavior of the $::scale property for the Gif external component has changed affecteing how single- and multi-frame gifs are scaled. If $::scale is true, the image is stretched to fit the control when the gif file is single frame, while the image bitmap is stretched to its equivalent *logical size* if the gif file is multi-frame.

Page Pane

The Paged Pane window class control now supports the $alltabcaptions property – this contains the values of the $pagename property of the pages.

Color Picker

The behavior of the Color Picker control has been improved it is used in an entry field to enter color values. When the palette opens in RGB mode, the entry field now has the focus, and selects all of the text. As you type, the color value updates. You can press return in one of the entry fields to set the color property, or Escape to close the color picker.

In addition, when the focus is on one of the normal selection arrays, pressing an arrow key removes the mouse capture, so you can navigate using the arrow keys.

Control Characters

You can now specify that control characters are visible in window class Edit, Multi-line edit, Combo box, Data grid, and String grid controls (in the Omnis runtime only). A new library preference $showcontrolcharacters has been added to enable this for the whole library, or you can set the property for individual controls of those types: you can set the property in the Property Manager or in your code.

Do $cobj.$showcontrolcharacters.$assign(kTrue)

Do $clib.$prefs.$showcontrolcharacters.$assign(kTrue)

When set to true, control characters are drawn using a suitable symbol, rather than space which is the default (when the property is false). Control characters are characters with a value less than Space (with the exception of carriage return for window controls which use CR as a line delimiter) and Del (0x7f).

Title Bar

On macOS High Sierra or above, end users can now double-click on a window title bar to Zoom/Minimize the window, but only if the window has Zoom/Minimize buttons.

Mouse Events

A new property called $movebehind has been added to window classes that allows you to control whether or not to allow mouse move events on fields in windows other than the top window: the property on the Action tab in the Property Manager.

By default $movebehind is set to kTrue and will allow mouse move events to be processed in other windows when the window is the top window, e.g. evMouseEnter and evMouseLeave. Set the property to kFalse to turn off this behavior for the window.

In previous versions, only evMouseEnter and evMouseLeave in a complex grid in a window that was not top were allowed. To revert to the legacy behavior, add an entry called "oldMouseMoveBehindBehaviour" to the "defaults" group in the config.json file and set it to true.

Dialog Windows

In previous versions, dialog windows were not drawn with a shadow at Runtime. Omnis Studio 10 now turns shadow effect on for all windows except docked toolbars, but the old behavior is configurable.

To turn on the old behavior, add an entry called "oldWindowShadowBehaviour" to the "macOS" group in config.json and set it to true. The legacy behavior disables the shadow effect for certain configurations of windows, e.g. dialog windows with no title, windows with no frame and title, and for floating toolbar windows.

$container

$container now returns the window instance for controls (including subwindows) at the top level: in previous versions this was not available for window class controls, only JavaScript controls. For example, you can use $cinst.$container() to refer to the outer window instance when executed in a subform window.

If you have a loop in your code that steps through from a window class control up the container hierarchy the final container will be the window, so now you will need to test if the container is a window, e.g. If itemref.$container().$ref.$classtype=kWindow, then Break to end of loop.

Encryption

Blowfish

There is a new property, $padding, in the Blowfish object to allow you to specify the type of padding to use when encrypting data.

- **$padding**
  A kBlowFishPadding... constant that indicates the type of padding to use when encrypting or expect when decrypting (default kBlowFishPaddingNone). A value other than kBlowFishPaddingNone is ignored if you specify a length header.

Valid values of the padding constant are kBlowFishPaddingNone (use or expect no padding) and kBlowFishPaddingPKCS5 (use or expect PKCS5 padding).

The presence of PKCS5 padding allows the code decrypting the data to correctly restore its length, without requiring the non-standard length header. This allows the BlowFish object to be used to encrypt data to be passed to applications other than Omnis – these applications (assuming they have the key) can decrypt the data and set its length correctly.

Report Programming

Save PDF on Print Preview

The Page Preview window now has a Save PDF button. You can control whether this button is present for user reports, using the root device preference

$reporttoolbarpagepreview: there is a new constant kRBsavePDF that controls whether the button is present.

Tabs in Reports

There is a new entry in config.json to handle tabs in reports better, so they are displayed properly on reports, e.g. on the preview thumbnail in page preview. The new option "replaceTabsInRTFwithSpacesWhenAddingToReport" is in a new group called "docview". This can be a value from zero to 32 inclusive, the default is 2. Zero means leave the text unchanged (the behavior prior to this update). 1-32 means replace each tab character found in the text with 1-32 spaces, when adding the text to a print job.

FileOps

FileOps Error Codes

The error codes that are generated by general file operations in Omnis and by the FileOps methods (kerr.. and kFileOps..) have been rationalized. The following general error code constants have been added:

kerrAlreadyExists	1215
kerrDiskRead	1001
kerrDiskWrite	1002
kerrEOFRead	1207
kerrEOFWrite	1208
kerrNotFound	1201
kerrNotOpen	1202

The codes for the following kFileOps... error constants have changed:

kFileOpsPermissionDenied	101203
kFileOpsDiskFull	101206
kFileOpsFileNotOpen	101202
kFileOpsEndOfFile	101207
kFileOpsFileNotFound	101201
kFileOpsFileLocked	101205
kFileOpsAlreadyExists	101215

All other FileOps codes remain unchanged.

Omnis VCS

Conversion

For Studio 10, you must create a new VCS repository due to the changes in the Omnis language syntax resulting from the new code editor. You are advised to open and convert your library, then check the conversion logs to look at any possible issues in your code (any conversion issues are shown in the Find and Replace log, and written to a log file in the 'conversion' folder in the logs folder). Then when you are satisfied your library and its code are OK, you can check the classes in your library into the new VCS repository.

Check Out from Find & Replace Log

You can check out a class from the VCS directly from the Find & Replace log window, assuming the class is not already checked out. You can select a line or multiple lines in the Find log, right-click on the selection, and select the Check Out option. You will be prompted to log onto the VCS if required and the Check-out dialog will de displayed containing the selected classes ready to be checked out.

Omnis IDE

Main menu and Themes

The $alwaysshowmainmenu property now works when not using themes (this property is only relevant for Windows Vista or 7).

Commands

Working Message

The *Working Message* command has a new option "Do not auto close". If specified, the command opens a modal working message dialog that does not automatically close when the method ends. While the message is open, subsequent working message calls with this option increment the message and ignore the other parameters.

The new option allows a working message to stay visible while calling one or more asynchronous methods, e.g. when calling a method in an HTML control via OBrowser, or when waiting for one or more callbacks from workers.

You must call *Close working message* to close the message. In a development version, although the message is modal, Omnis leaves the View menu enabled, so if you do not call *Close working message*, so you can open an IDE window, such as the Studio Browser, which closes the working message.

Functions

Binary functions

URL encoding and padding options have been added to the bintobase64() and binfrombase64() functions.

bintobase64()

bintobase64(vData[,bURLEncoding=kFalse,bAddPadding=kTrue])

Pass *bURLEncoding* as kTrue to use the URL-safe form of base64.

Pass *bAddPadding* as kFalse to exclude the padding from the base64 (the one or two = characters appended to the end of the encoded data). Typically, this would be passed as kFalse when using URL-safe encoding.

binfrombase64()

binfrombase64(vData[,bURLEncoding=kFalse,bExpectPadding=kTrue])

Pass *bURLEncoding* as kTrue to decode the URL-safe form of base64.

Pass *bExpectPadding* as kFalse to not expect any padding in the base64. Typically, this would be passed as kFalse when using URL-safe encoding.

byteget()

byteget(binary,byteNumber)

Returns the value (0-255) of the byte at the specified *byteNumber* in *binary*. Returns -1 if an error occurs.

sys()

sys(239)

sys(239) returns true if the Startup method in the library has finished, or false if not.

sys(240)

sys(240) returns a string that identifies the current monitor configuration. You could use this in your code to branch according to the monitor configuration of end users.

sys(241)

sys(241) returns a copy of the find and replace log list from $findandreplace.

mouseover()

There is a new constant for the mouseover() function, kMComplexGridRow, that returns the row of a Complex grid the pointer is over.

sleep()

The sleep() function will pause execution for a minimum of the specified number of milliseconds – the actual delay can be longer in the multi-threaded server if another stack is executing its time slice when the sleep delay expires. As soon as you execute sleep() in the multi-threaded server, other waiting stacks can run.

Notation

Find and Replace

Some improvements have been made to $findandreplace notation. Passing #NULL as the replace string performs a find all rather than a replace all in the class. In addition, sys(24f1) returns a copy of the find and replace log list.

Appendix A

Obsolete Commands

Some of the obsolete commands have been removed from this version: these commands were marked with “OBSOLETE COMMAND” and appeared in the ‘Obsolete commands…’ group in the Command list in pre-Studio 10.x versions. The converter in Studio 10.x will comment out these commands wherever they appear in your code, and a record of the conversion process is added to a log file in the /logs/conversion folder.

* The *Call method OBSOLETE COMMAND* is not commented out, but is converted to *Do code method* using the same parameter as the old command.

The *Translate input/output* command is now obsolete and will be commented out in your converted code.

Autocommit OBSOLETE COMMAND

Begin SQL script OBSOLETE COMMAND

Build list from select table OBSOLETE COMMAND

Build list of event recipients OBSOLETE COMMAND

Call method OBSOLETE COMMAND * (converted to Do code method)

Cancel event recipient OBSOLETE COMMAND

Cancel publisher OBSOLETE COMMAND

Cancel subscriber OBSOLETE COMMAND

Close client import file OBSOLETE COMMAND

Close cursor OBSOLETE COMMAND

Commit current session OBSOLETE COMMAND

Declare cursor for OBSOLETE COMMAND

Delete client import file OBSOLETE COMMAND

Describe cursors OBSOLETE COMMAND

Describe database OBSOLETE COMMAND

Describe results OBSOLETE COMMAND

Describe server table OBSOLETE COMMAND

Describe sessions OBSOLETE COMMAND

Disable automatic publications OBSOLETE COMMAND

Disable automatic subscriptions OBSOLETE COMMAND

Disable receiving of Apple events OBSOLETE COMMAND

Enable automatic publications OBSOLETE COMMAND

Enable automatic subscriptions OBSOLETE COMMAND

Enable receiving of Apple events OBSOLETE COMMAND

End SQL script OBSOLETE COMMAND

Execute SQL script OBSOLETE COMMAND

Fetch current row OBSOLETE COMMAND

Fetch first row OBSOLETE COMMAND

Fetch last row OBSOLETE COMMAND

Fetch next row OBSOLETE COMMAND

Fetch previous row OBSOLETE COMMAND

Get SQL script OBSOLETE COMMAND

Logoff from host OBSOLETE COMMAND

Logon to host OBSOLETE COMMAND

Make file class from server table OBSOLETE COMMAND

Make schema from server table OBSOLETE COMMAND

Map fields to host OBSOLETE COMMAND

Open client import file OBSOLETE COMMAND

Open cursor OBSOLETE COMMAND

Open desk accessory OBSOLETE COMMAND

Perform SQL OBSOLETE COMMAND

Prepare current cursor OBSOLETE COMMAND

Prompt for event recipient OBSOLETE COMMAND

Prompt for word server OBSOLETE COMMAND

Publish field OBSOLETE COMMAND

Publish now OBSOLETE COMMAND

Quit cursor(s) OBSOLETE COMMAND

Reset cursor(s) OBSOLETE COMMAND

Retrieve rows to file OBSOLETE COMMAND

Rollback current session OBSOLETE COMMAND

Send core event OBSOLETE COMMAND

Send core event with return value OBSOLETE COMMAND

Send database event OBSOLETE COMMAND

Send finder event OBSOLETE COMMAND

Send to publisher OBSOLETE COMMAND

Send word services event OBSOLETE COMMAND

Server specific keyword OBSOLETE COMMAND

Set batch size OBSOLETE COMMAND

Set character mapping OBSOLETE COMMAND

Set client import file name OBSOLETE COMMAND

Set current cursor OBSOLETE COMMAND

Set current session OBSOLETE COMMAND

Set database version OBSOLETE COMMAND

Set event recipient OBSOLETE COMMAND

Set hostname OBSOLETE COMMAND

Set password OBSOLETE COMMAND

Set publisher options OBSOLETE COMMAND

Set SQL blob preferences OBSOLETE COMMAND

Set SQL script OBSOLETE COMMAND

Set SQL separators OBSOLETE COMMAND

Set subscriber options OBSOLETE COMMAND

Set transaction mode OBSOLETE COMMAND

Set username OBSOLETE COMMAND

SQL: OBSOLETE COMMAND

Start session OBSOLETE COMMAND

Subscribe field OBSOLETE COMMAND

Subscribe now OBSOLETE COMMAND

Translate input/output

Use event recipient OBSOLETE COMMAND