

What's New in Omnis Studio 10.2

Rev 31315

Omnis Software

November 2021

54-112021-01

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 2021. All rights reserved.

Portions © Copyright Microsoft Corporation.

Regular expressions Copyright (c) 1986,1993,1995 University of Toronto.

© 1999-2021 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>)

© 2001-2021 Python Software Foundation; All Rights Reserved.

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

Omnis® and Omnis Studio® are registered trademarks of Omnis Software.

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 © 1995-2021 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) 1996-2021, The PostgreSQL Global Development Group

Portions Copyright (c) 1994, The Regents of the University of California

Oracle, Java, and MySQL 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	6
SOFTWARE SUPPORT, COMPATIBILITY AND CONVERSION ISSUES.....	7
WHAT'S NEW IN OMNIS STUDIO 10.2 REV 31315	12
APPLE M1 & MACOS MONTEREY SUPPORT.....	12
THE OMNIS ENVIRONMENT.....	13
JAVASCRIPT REMOTE FORMS.....	14
REPORT PROGRAMMING.....	14
FUNCTIONS.....	14
WHAT'S NEW IN OMNIS STUDIO 10.2 REV 30204	15
LIST PROGRAMMING.....	15
WINDOW PROGRAMMING.....	15
OMNIS DATA BRIDGE.....	15
ORACLE DAM.....	16
WHAT'S NEW IN OMNIS STUDIO 10.2 REV 29818	17
JAVASCRIPT COMPONENTS.....	17
JAVASCRIPT REMOTE FORMS.....	17
OMNIS ENVIRONMENT.....	17
LIBRARIES.....	18
WINDOW COMPONENTS.....	18
WHAT'S NEW IN OMNIS STUDIO 10.2 REV 29538	19
JAVASCRIPT COMPONENTS.....	19
JAVASCRIPT REMOTE FORMS.....	20
OMNIS ENVIRONMENT.....	20
WINDOW PROGRAMMING.....	21
WINDOW COMPONENTS.....	21
REPORT PROGRAMMING.....	22
DEPLOYMENT TOOL.....	22
OMNIS GRAPHS.....	22
EXTERNAL COMPONENTS.....	23
WHAT'S NEW IN OMNIS STUDIO 10.2 REV 28632	24
JAVASCRIPT REMOTE FORMS.....	24
JAVASCRIPT COMPONENTS.....	24
CODE EDITOR.....	25
LIBRARIES.....	26
OMNIS ENVIRONMENT.....	26
WINDOW COMPONENTS.....	26
FUNCTIONS.....	26
WHAT'S NEW IN OMNIS STUDIO 10.2	28
JAVASCRIPT COMPONENTS.....	30
JAVASCRIPT FORMS.....	51
METHOD EDITOR.....	54
MULTIPROCESS SERVER.....	62
WINDOW COMPONENTS.....	69
WINDOW PROGRAMMING.....	82
OMNIS LIBRARIES.....	87
OMNIS ENVIRONMENT.....	88

LOCALIZATION	91
REPORT PROGRAMMING	92
OW3 WORKER OBJECTS	92
WEB SERVICES	100
OBJECT ORIENTED PROGRAMMING	101
JSON COMPONENTS	101
COMMANDS.....	101
FUNCTIONS	102
OJSON	104
JAVASCRIPT API	104
IMPORT/EXPORT	105
OMNIS VCS	105
DEPLOYMENT.....	105
OMNIS DATAFILE MIGRATION.....	106
EXTERNAL COMPONENTS.....	106
WHAT'S NEW IN OMNIS STUDIO 10.1	107
CODE EDITOR	109
SQL WORKER LISTS	118
JAVASCRIPT REMOTE FORMS	121
JAVASCRIPT COMPONENTS.....	124
COMMANDS.....	131
WINDOW CLASSES & COMPONENTS.....	131
FUNCTIONS	136
OMNIS ENVIRONMENT	137
LIBRARIES AND CLASSES.....	139
JAVASCRIPT WORKER.....	140
REMOTE DEBUGGER	140
OMNIS DATAFILE MIGRATION.....	140
LIST PROGRAMMING.....	141
OBJECT CLASSES	141
FILE CLASSES	141
WEB SERVICES	141
REPORT PROGRAMMING	142
LOCALIZATION	143
JSON CONTROL EDITOR.....	144
OJSON	144
OW3 WORKER OBJECTS	144
DEPLOYMENT.....	145
OMNIS VCS	145
EXTERNAL COMPONENTS.....	146
WHAT'S NEW IN OMNIS STUDIO 10.0	147
METHOD EDITOR.....	148
ACCESSIBILITY	172
JAVASCRIPT REMOTE FORMS	175
JAVASCRIPT COMPONENTS.....	179
REMOTE DEBUGGER	197
REMOTE OBJECTS	205
WEB AND EMAIL WORKER OBJECTS.....	207
JSON COMPONENTS	215
REPORT PROGRAMMING	216
LIBRARIES.....	216
COLOR THEMES AND APPEARANCE	217
STUDIO BROWSER	217
FIND AND REPLACE	218

LOCALIZATION	218
DEPLOYING YOUR WEB & MOBILE APPS	221
SQL PROGRAMMING	222
OMNIS PROGRAMMING	222
WEB SERVICES	223
WINDOW CLASSES & COMPONENTS	223
ENCRYPTION	227
REPORT PROGRAMMING	227
FILEOPS	228
OMNIS VCS	228
OMNIS IDE	229
COMMANDS	229
FUNCTIONS	229
NOTATION	230
APPENDIX A	231
OBSOLETE COMMANDS	231

About This Manual

This document describes the new features and enhancements in Omnis Studio 10.2 Revision 30204 (as well as revisions 29818, 29538 and 28632), as well as Omnis Studio 10.1 and 10.0.

Please see the **Readme.txt** file for details of bug fixes and any release notes for Omnis Studio 10.2 Rev 30204.

Software Support, Compatibility and Conversion Issues

The following section contains issues regarding software support, compatibility and conversion in the Omnis Studio 10.2 Rev 30204 patch release and other Omnis Studio 10.2 releases.

Serial Numbers and Licensing

You will require a new serial number to run Omnis Studio 10.2. Contact your local sales office to buy a license or obtain an upgrade serial number under your current support program, or go to our website: www.omnis.net

Library and Datafile Conversion

IMPORTANT: IN ALL CASES, YOU SHOULD MAKE A SECURE BACKUP OF ALL OMNIS LIBRARIES AND OMNIS DATAFILES BEFORE OPENING THEM IN OMNIS STUDIO 10.2.

Converting 10.0.0 Libraries

***** IMPORTANT NOTE: *****

ONCE A STUDIO 10.0.x LIBRARY HAS BEEN OPENED WITH OMNIS STUDIO 10.1 or 10.2 IT CANNOT BE OPENED WITH STUDIO 10.0.x.

Converting 8.x or earlier Libraries

Omnis Studio 10.2 will convert existing version 8.1.x, 8.0.x, 6.1.x, 6.0.x and 5.x libraries – THE CONVERSION PROCESS IS IRREVERSIBLE.

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

Omnis Studio on macOS

macOS Monterey Support

Omnis Studio 10.2 Rev 31315 is certified to run on macOS 12 Monterey.

Running 10.2 on Big Sur

Omnis Studio 10.2 is certified to run on macOS 11 Big Sur from Omnis Studio 10.2 Rev 28632 patch onwards.

Studio 10.2

We released Omnis Studio 10.2 November 10, 2020, just before the official release of Big Sur. It is possible that there will be compatibility issues, including some drawing anomalies, when running the release version of Omnis Studio 10.2 on Big Sur. Drawing support has been changing with each beta release of Big Sur that Omnis engineering has been working with, but we have addressed issues now that Big Sur is released, and the fixes were in the Studio 10.2 Rev 28632 patch release.

Studio 10.1

Omnis Studio 10.1 will operate on Big Sur, but it will not be a certified or supported configuration. We therefore recommend that you upgrade to Omnis Studio 10.2 as soon as possible.

Studio 8.1

Big Sur is a large step forward in architecture and developers should note that older applications such as Studio 8.1 cannot simply be adjusted for immediate use on this new version of macOS.

We therefore recommend upgrading to Omnis Studio 10.2 if you believe there is a business need or individual use case for you or your end users to use macOS Big Sur.

Window Refresh on macOS

The `preventUpdateWithNoRefreshOn` config.json item was introduced in Studio 10.1 Rev 29237 to handle window refresh on Big Sur, but this item was added to and modified for Studio 10.2 Rev 30204. In Studio 10.1 Rev 29237 this property was only applicable when running on Big Sur but now applies to all versions of macOS, i.e. from 10.14, to 11 and 12.

When `preventUpdateWithNoRefreshOn` is set to true and a window has set `$norefresh` to `kTrue` then this will prevent changes to the window hierarchy, e.g. adding fields, from causing a redraw to screen. The window changes will then be applied when `$norefresh` is set to `kFalse`.

This is a hidden property that needs to be explicitly added to the config file. It is set to false by default.

Node.js

Node.js is used in a number of features in Omnis Studio including Remote Debugging and the JavaScript Worker.

The version of Node.js released with the macOS version of Studio 10.2 is now V16.6.1 which supports node running natively on macOS on M1 machines.

Gif Control

The GIF control has been removed from Omnis Studio 10.2 and is no longer supported in this or future releases. You should find an alternative method to display GIF images (a third-party external GIF control) or convert your images to a supported image format such as PNG or JPG.

Default Printer (Windows)

The way the default printer on Windows is returned changed in the initial release of Omnis Studio 10.2 and this has caused a few issues with some converted applications. A workaround is to revert to the previous behavior by setting the new entry `"useLegacyDefaultPrinter"` to true in the "windows" section of config.json.

Rebuilding External Components

All external components will need to be rebuilt to run with Omnis Studio 10.2 or above using the source files from in the Ext Comp SDK accompanying this version.

macOS External Components

For Omnis Studio 10.2 or above we now use a newer version of Apple's SDK to build Omnis and our external component library and this requires a newer version of the SDK and Xcode when building components for 10.2.

To be compatible with the Studio 10.2 SDK and later your component needs to use at a minimum the macOS 10.14 SDK with a minimum deployment target of 10.11. This requires a minimum of Xcode 10.2 on macOS 10.14.4.

The C++ Language Dialect must be set to support C++11 as a minimum, and the C++ Standard Library set to libc++ with C++11 support accordingly.

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 the \$active property 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.

Drag and Drop

Support for dragging and dropping *operating system files* and *file data* (in the thick client) has been combined and simplified providing more control in your event handling code. As a consequence there may be some compatibility issues, but these are outlined later in this document.

IE 11 Support

Omnis Studio 10.2 or above *does not support* IE 11 (or earlier) for the JavaScript Client. Microsoft will end support for the desktop version of IE 11 in November 2020.

Open SSL

Omnis Studio 10.2 supports OpenSSL 1.1.1 so you should update your copy. OpenSSL is used to provide SSL in the OW3 Workers.

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), which has an impact on the command *Enclose exported text in quotes (Enable)*. 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".

Method Editor: Code Conversion

IMPORTANT NOTE FOR PRE-STUDIO 10.x USERS: There has been a major rewrite of the code editing part of the Method Editor (in Studio 10.0), which means you can now enter Omnis commands and code using freetype. Due to these major enhancements, there has been several enhancements or changes in the Omnis *programming language* and *command syntax*. Therefore, when you convert an Omnis library to Studio 10.x or above (e.g. from Studio 8.x or earlier), your Omnis code will be converted to the new syntax as part of the library conversion process.

Once you convert your library and start using the new free-type Code Editor in Studio 10.x or above, ***you cannot revert back to the old editor in Studio 10.x (or above)***: that is, the old interface for modifying methods, using point-and-click, *has been removed from Studio 10.x*.

See 'Library Conversion' under the Code Editor section (under Studio 10.0 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 10.1, or above, 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.x 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 superseded 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 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.\$methodname()** rather than **OWEB.\$methodname()**.

The OWEB methods affected are:

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

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 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 10.2 Rev 31315

The following enhancements have been added to Omnis Studio 10.2 Rev 31315. Please see the Readme.txt file accompanying the release for details of bug fixes in this release. The following features have been added to Studio 10.2 Rev 31315:

- ❑ **Apple M1 & macOS Support**
this release supports application development and deployment using Omnis Studio 10.2 on macOS running natively on M1 (arm64) and Intel (x86_64) based Apple computers; plus this release is certified to run on macOS 12 Monterey
- ❑ **Main Window Resize Message**
A new task message \$mainresized has been added, which is called when the main Omnis window has been resized on the Windows platform
- ❑ **JavaScript Forms**
any subform in the inheritance hierarchy can now have different breakpoints to their superclass
- ❑ **Reports**
the HTML Icon (link) external component now has the \$keepaspectratio property to ensure the icon is displayed correctly

Apple M1 & macOS Monterey Support

This release on macOS is a 'Universal build' meaning that it will run natively on M1 (arm64) and Intel (x86_64) based Apple computers. There are a number of enhancements that support Omnis Studio running on M1 based Macs or macOS 11+. Plus this release is certified to run on macOS 12 Monterey.

Toolbars

There is a new config.json item **useToolbarStyleExpanded** to allow the legacy expanded toolbar style instead of the default (typically unified).

The **useToolbarStyleExpanded** config item has been added to the 'macOS' section of the config.json file and only applies to macOS 11 and later. When set to true, the window toolbar style will use the legacy expanded style, i.e. toolbars sit under the window title. By default, this is false and toolbars will use the new automatic style on macOS 11 and later, i.e. toolbars are unified and to the right of the window title.

sys(8)

The sys(8) function returns MACARM when Omnis is running on an M1 (arm64) based Mac.

Building macOS Universal Components

In order to build macOS Universal components you need to use Xcode 12 or above. The Omnis resource compiler is now a macOS Universal binary and should replace the version in the Xcode tree at:

```
/Applications/Xcode.app/Contents/Developer/Tools/omnisrc64.app
```

The macOS resource compiler now expects a UTF-8 encoding for the strings it reads from a resource file. It will write a message to the build log if a string is encoded incorrectly. Although a message is written to the build log, the build continues,

however, you get a warning about an incomplete Localizable.strings file, and this is the indicator that you need to look at the build log.

This is only an issue for a small number of text resources, e.g. extended ASCII codes such as the Copyright symbol or the £ character.

A component which is currently using the existing Intel only 10.2 SDK with Xcode 11 should be compatible with Xcode 12. Once opened with Xcode 12 this should build out a Universal binary version (a fat x86_64 / arm64 binary).

The important Xcode build settings are:-

Architectures

Should be \$(ARCHS_STANDARD_64_BIT)
or \$(ARCHS_STANDARD) /
Standard Architectures (Apple M1, Intel).

Build Active Architecture Only

Should be No in order to build out both supported architectures. Typically this will be No for Deployment and Yes for Development.

The generic and jsgeneric components provide a good starting template for a project.

The component library and any third-party libraries that a component uses **MUST** also be Universal to build out a Universal component.

To check the architecture included in a binary use the lipo command from the Terminal, e.g.

```
% lipo -info /Users/macminisilicon/Downloads/OSX-SDK-10.2-31054-
Beta/_OSXUnicode/generic.u_xcomp/Contents/MacOS/generic
Architectures in the fat file: /Users/macminisilicon/Downloads/OSX-SDK-10.2-
31054-Beta/_OSXUnicode/generic.u_xcomp/Contents/MacOS/generic are: x86_64
arm64
```

For a binary to load natively on both Intel and M1 machines it must include both the x86_64 and arm64 architectures.

Note. An Intel only component cannot be loaded by the Universal version of Omnis Studio running natively on Apple M1 machines.

Rebuilding External Components

If you are upgrading from Studio 10.1 or before, you will need to rebuild all external components to run with Omnis Studio 10.2 or above using the source files from the Ext Comp SDK accompanying this version.

The Omnis Environment

Main Window Resize Message

A new task message **\$mainresized** has been added, which is called when the main Omnis window has been resized on the Windows platform (it does not apply on macOS).

\$mainresized has two parameters, pWidth and pHeight, which are the dimensions of the available area of the main window (excluding any docking areas if present). When the main window is minimized the parameters are both zero.

In addition, there are new sys functions sys(251) and sys(252) that return the width and height of the available area of the main window, respectively.

Code Editor

Bad name detection has been added to the Code Editor, so bad notation names are detected while entering code rather than handling this through automatic retokenization using double slashes.

There is a new item 'badNotationNameIsSyntaxError' in the 'ide' section of config.json, which defaults to true, enabling the new behavior. Set this to false to restore the previous behavior.

JavaScript Remote Forms

Layout Breakpoints

It is no longer required that all subforms within the inheritance hierarchy of a set of Remote forms have the same layout breakpoints. In other words, a subform can now have different layout breakpoints to its superclass.

Report Programming

HTML Icon (Link)

The HTML Icon (Link) external component (available for report classes) did not always display the icon correctly in previous versions. Therefore, the component now has the \$keepaspectratio property which needs to be set to kTrue for the icon to draw properly (the default is kFalse to maintain backwards compatibility).

Functions

mouseover()

The constant kMScreenPos has been added for use with the mouseover() function. mouseover(kMScreenPos) returns a row with 2 columns, h and v, which are the horizontal and vertical position of the pointer (mouse) in screen coordinates, respectively.

sys(251)

Returns the width of the available area of the main Omnis application window.

sys(252)

Returns the height of the available area of the main Omnis application window.

What's New in Omnis Studio 10.2 Rev 30204

The following enhancements have been added to Omnis Studio 10.2 Rev 30204. Please see the Readme.txt file accompanying the release for details of bug fixes in Studio 10.2 Rev 30204.

List Programming

List Column Calculations

A new library preference \$clib.\$prefs.\$validcolumninbadrowisnull has been added. If true, non-existent list columns in calculations evaluate to #NULL rather than an empty character string. This allows for expressions like myList.col or myList.10.col where the list line does not exist, perhaps because the list is empty.

Window Programming

Folders in Operating System Drag and Drop

Due to issues dropping folders when dragging items from Omnis onto the operating system, folders are now included in the list of dropped objects, with a size of zero.

Using Non-TrueType fonts for Background Objects

A new config.json item has been added to allow you to use non-TrueType fonts for window background objects.

The config.json item 'backgroundObjectsMustUseTrueTypeFont' has been added to the 'windows' section of config.json. If true (the default) TrueType fonts must be used. When false, you can use non-TrueType fonts for background objects, but note that in some situations, e.g. in drag bitmaps, the text may not draw.

Omnis Data Bridge

ODB Encryption

The \$odbencrypt session property has been added. If kTrue (the default) ODBC Data Bridge connections use end-to-end encryption. Improved network performance can be achieved by disabling encryption. The ODBC Data Bridge uses the value that is in effect when \$logon() is called, i.e. if kTrue when \$logon() is called, fetch results will still be encrypted for the duration of the connection even if \$odbencrypt is subsequently cleared.

Note that you do not need to update the ODBC Data Bridge to use this feature, since it automatically recognizes encrypted and non-encrypted data, and responds in kind.

Oracle DAM

RPC Methods

The `$rpcprocedures()`, `$rpcparameters()`, `$rpcdefine()` and `$rpc()` methods have been added to the Oracle DAM.

`$rpc()` executes a PL/SQL `begin... end` statement block that calls the stored procedure or function. Operation is as described in the *SQL Programming* chapter with one exception. When binding single-column `SELECT` tables, it is necessary to pass the required list column numbers along with the parameter definitions. To do this, Omnis makes use of column 5 of the list returned by `$rpcparameters()`. For example:

```
Do cStat.$rpcparameters('credit') Returns #F
  Do procList.$define()
  Do cStat.$fetch(procList,kFetchAll)      ## returns 4 rows
  Do procList.3.5.$assign(1)      ## Assign the list column number to 1
  Do procList.4.5.$assign(2)      ## Assign the list column number to 2
Do cSess.$rpcdefine('credit',procList) Returns #F
  Do lCreditList.$define(lName,lBalance)
Do cStat.$rpc('credit',1,10,lCreditList,lCreditList) Returns #F
```

The additional values assigned to `procList` correspond to the column numbers that would otherwise be passed via the `$plsqli()` method.

You can also call a stored function using the `$rpc()` method and the return value will be written to the statement object's `$rpcreturnvalue` property. For example:

```
Begin statement
  Sta: CREATE OR REPLACE FUNCTION test_function
  Sta: RETURN VARCHAR2 IS
  Sta: BEGIN
  Sta: RETURN 'This is being returned from a function';
  Sta: END test_function;
End statement
Do cStat.$execdirect() Returns #F
Do cStat.$rpcparameters('test_function') Returns #F
  Do procList.$define()
  Do cStat.$fetch(procList,kFetchAll)
  Do cSess.$rpcdefine('test_function',procList) Returns #F
Do cStat.$rpc('test_function') Returns #F      ## now check the value of
  $rpcreturnvalue
```

`$rpc()` is limited to calling a single stored procedure or function. To execute more complex PL/SQL constructs, you can continue to use the `$plsqli()` method.

What's New in Omnis Studio 10.2 Rev 29818

The following enhancements have been added to Omnis Studio 10.2 Rev 29818. Please see the Readme.txt file accompanying the release for details of bug fixes in Studio 10.2 Rev 29818.

JavaScript Components

Virtual Keyboard & \$negallowed

The \$inputtype for JS Edit fields is no longer set to 'number' if \$negallowed is true, as these don't guarantee the presence of a minus key. This may mean that some situations which previously showed a numeric keyboard no longer will. You should note that a touch device's virtual keyboard is more likely to use a numeric keyboard if \$negallowed is false.

JavaScript Remote Forms

\$construct Row

A 'clientPlatform' column has been added to the \$construct Row parameter for remote forms. This denotes the platform on which the client is running, and returns one of the following strings: 'Windows', 'macOS', 'Linux', 'iOS', 'Android' or 'Unknown'.

Omnis Environment

Code Assistant

There is a new Boolean item **listShowsNamesFirst** in the 'codeAssistant' section of config.json, to show method names before attributes in Code Assistant lists (it defaults to true). When true, names occur in the Code Assistant list before attributes etc that start with \$. When false, the list order is the same as in previous versions, where \$ entries typically occur before names.

Method Editor

There is a new item "methodeditorfadealpha" (value 0-255) in the "IDEmethodEditor" section of appearance.json to allow you to set the fade level of the method editor when editing a variable value in the debugger variable panel.

DB view in Query Builder

An option has been added to the 'Other' menu in the Query Builder to create a DB view.

Class Comparison Tool

You can now filter the list of classes in the Class Comparison tool on keypress for revisions.

Hub Samples

There is a new sample app showing the use of \$userworker in SQL Worker Lists in the Samples section of the Hub in the Studio Browser.

Libraries

JSON Export

The reporting of conflicts in JSON Export has been improved. Note that the conflict detection process uses the modify date of each file in the JSON tree for the class, so if a date has changed a conflict will still be reported even if the file contents have not changed. Note also that this means conflicts will be reported (if overwrite conflicts is off) when you first export a library with this updated version of the JSON export.

Window Components

Entry Fields

A new property \$showellipsis has been added to the fat client Single Line Entry Field (only applies when field is read-only, i.e. the data is not being edited). If true, an ellipsis is shown at the end of truncated text in the field if the text is too long to be displayed (this only applies when the control is read-only, \$horzscroll and \$righttoleft are both kFalse, \$align is kLeftJst and \$passwordchar is not set).

Note that the edit field always includes at least the first character of the text, so very narrow edit fields will sometimes show truncated text, but in most cases this will not be apparent.

What's New in Omnis Studio 10.2 Rev 29538

The following enhancements have been added to Omnis Studio 10.2 Rev 29538. Please see the `Readme.txt` file accompanying the release for details of bug fixes in Studio 10.2 Rev 29538.

JavaScript Components

Position Assistance

When positioning objects in the center of a remote form the Position Assistance feature now uses the center of *the current layout breakpoint*, not the center of the remote form design window, as in previous versions.

Hot Control Properties

Several JS controls will now use their “hot” colors *when they have the focus*, and not just when the pointer is over them, as in previous versions. This enhancement applies to the “hot” properties for the following controls: Nav menu, Split button, Hyperlink, Nav bar, Tab bar, and Trans button.

Border Radius

The `$borderradius` property has been added to the Date Picker and Popup Menu controls allowing you to apply a border radius to these controls.

Paged Pane

The `$fieldstyle` property has been added to the Paged Pane control for the JS Client allowing you to apply a style to the control.

Data Grid

Assigning Colors

Using `kEscColor` with the `style()` function to change the color of items in a Data grid has been improved.

The parameters for `style()` can now be any HTML color string, such as `"#FF0000"`. For client methods that execute on the client, the color parameter *must be a literal string* and therefore enclosed in double quotes. For example, `style(kEscColor,"#FF0000")`, or `style(kEscColor,"rgba(0,0,255,0.5)")`.

Omnis does not validate the HTML color syntax, so you should check the syntax is correct to avoid runtime errors.

Frozen Columns

The `$frozencolumns` property in Data Grids can now be assigned at runtime.

JavaScript Remote Forms

Timeouts in Remote Tasks

The `$ondisconnected` remote task method is now shown in the built-in methods list in the method editor and Interface Manager.

Omnis Environment

Find and Replace

When using Find and Replace, the found or replacement text is now highlighted in the Find and Replace log. The **Highlight Matches** option in the context menu for the log allows you to toggle the find or replace text highlighting (the default is on).

The color used for highlighting the found or replacement text is the *method line highlight color*, that is, the highlight color used when the Code Editor field does not have the focus. If the text occurs more than once, up to the first 16 occurrences in the log are highlighted.

Catalog

You can now drag variables and other items from the **Catalog** (F9/Cmnd-9) to the *Initial value* and the *Description* fields in the variable pane of the method editor: for this to work, the focus must be on the initial value or description field *before switching to the Catalog* to select the item.

Clipboard Commands for Fields

The clipboard menu items Cut, Copy, Paste, etc are now available for all entry fields in the Omnis environment, such as in the Method Editor, and for Entry fields in your apps when you Right-click/Option-click on the field.

Boolean Variable Values

The value of **Boolean** variables is now shown in a tooltip in the Code Editor when you hover over the variable. The "Show Empty Booleans" option has been added to the Debugger Options menu in the Code Editor to control whether empty Booleans are shown as Empty or No/False; the default is on, meaning that unset Booleans are shown as empty.

Tooltips

You can now specify the maximum width for tooltips, used within the Omnis IDE, e.g. in the Property Manager, and for window controls in all but a few special cases. The **maxWidth** setting in the 'tooltip' section of the `appearance.json` file specifies the width in pixels; it defaults to 0 meaning tooltips can be up to a third of the width of the screen or application window.

Notation Errors

A new item 'stricterNotationErrorChecks' has been added to the 'defaults' section of the `config.json` file. When set to true, certain unresolved name errors (from such notation in the form `$cinst.name` or `$ctask.name`) now result in a debugger (or runtime) error if `$clib.$prefs.$reportnotationerrors` is `kTrue`. The option defaults to false, so there is no change to behavior unless you enable the new option.

Trace Log

You can now copy selected lines from the **Trace log** to the clipboard using the Edit menu **Copy** command or Ctrl/Cmnd-C shortcut key.

Omnis Configuration File

Omnis Port

There is a new item **disableInRuntime** in the 'server' section of the config.json file to prevent the Omnis Server listening on its own port: this can be used to prevent firewall prompts when the Omnis Server is not required.

File associations and UAC

The items in the config.json file regarding file associations and UAC have been renamed, specifically regarding capitalization:

- UpdateFilesAssociations becomes **updateFileAssociations** (also the 's' was removed)
- NoAdmin becomes **noAdmin**
- HideStudiorgMessage becomes **hideStudiorgMessage**

Window Programming

Toast Messages

The `iStack` parameter in the `$showtoast` method has a new option **kToastStackCenter** to allow you to stack the toast messages in the center of the screen or application window.

Window Components

Complex Grids

The **\$extendedgridlines** property has been added to Complex grids. When set to `kTrue`, the grid lines of the final row extend to the base of the grid.

Key Events

Support for the Windows **VK_PAUSE** virtual key has been added to the `evKey` event. In this case, the `pSystemKey` event parameter has a value of 100 to signal the Pause button has been pressed.

Combo box

The **\$disablesearchonopen** property has been added to fat client Combo boxes, Data grids (applies to combo box columns), and toolbar Combo boxes.

If true, the automatic search is disabled, that is, the content of the combo box list *is not used* to populate the edit field based on the content of the edit field when the popup list is opened. For Data Grids, this property is used for columns with `$columnstype kDataGridComboPicker`.

Tab Pane

A new property **\$colortabselectedhighlightmacos** has been added to the fat client Tab Pane control to allow you to set the color of the active tab for tab panes on macOS only.

Window Resizing

The `evResized` event is now reported when a window with the `$edgefloat` property set to floating edges resizes due to the main Omnis application window being resized (this only applies on the Windows platform).

Report Programming

HTML Link Object

The `$tooltip` property has been added to the **HTML Link** report external component. This contains the tooltip used for the link specified by `$address` in Page Preview reports (tooltips will not work in PDF reports). It can contain expressions including square bracket notation.

Report Entry field

The `$linkaddress` and `$tooltip` properties have been added to the Report Entry Field. `$linkaddress` is the link address used by the Preview and PDF report destinations to provide a hyperlink. Note this provides similar functionality to the `$address` property of the HTML Link objects.

Deployment Tool

Build Folder

A new Go to menu has been added to the Deployment Tool that allows you to see the build folder in the system file explorer.

Omnis Graphs

High Resolution Charts

The `Graph2` component now draws charts in high resolution suitable for display on high resolution displays. In this case, charts are generated at twice the size and are displayed at the correct physical size on high resolution displays.

You can disable the new behavior by setting the new property `$disablehighresolution` to `kTrue` (the default is `kFalse` meaning high resolution charts are supported). If the client is running on a display that does not support high resolution, the property will be set to `kTrue` automatically, and you will not be able to change the value of the property.

External Components

There is a new Window Message to report the mouse wheel has been rotated.

WM_MOUSEWHEEL

The WM_MOUSEWHEEL message is sent to a window when the mouse wheel is rotated.

If the mouse is not captured, the message goes to the window beneath the cursor. Otherwise, the message goes to the window that has captured the mouse.

Parameters:

direction – Value of wParam. The high-order word indicates the distance the wheel is rotated. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user.

Returns:

An external component should return zero if it processes this message.

Notes:

On macOS

- 1) The direction value will be either 0, 1 or -1. No other values are supported.
- 2) Controls that have no scroll bars added will need to respond to WM_FLD_NEEDSWM_MOUSEWHEEL to receive a WM_MOUSEWHEEL message (see example)

Example:

```
// This is an example handing WM_MOUSEWHEEL
// Processing to be added to WNDPROC..
switch ( message )
{
    // controls with no scroll bars need to return 1L for the
    // WM_FLD_NEEDSWM_MOUSEWHEEL message to receive WM_MOUSEWHEEL
    case WM_FLD_NEEDSWM_MOUSEWHEEL: return 1L;
    //
    case WM_MOUSEWHEEL:
    {
        // Mouse wheel moved
        if ( wParam )
        {
            qbool lineUp = ((qshort)HIWORD(wParam)) >= 0;
        }
        // return 0 to indicate this control has processed this message.
        return 0L;
    }
}
```

#EXTCOMPLIBS file location

You can now copy the text from the #EXTCOMPLIBS file location field, plus the field will auto-scroll and displays a tooltip, making the file location more visible.

What's New in Omnis Studio 10.2 Rev 28632

Omnis Studio 10.2 was released in November 2020. This patch release provides some important fixes to fully support 10.2 on macOS Big Sur, plus some other bug fixes and minor enhancements.

The following enhancements have been added to Omnis Studio 10.2 Rev 28632. Please see the Readme.txt file accompanying the release for details of bug fixes in Studio 10.2 Rev 28632.

JavaScript Remote Forms

Remote Form Design

The Remote form Web Preview design mode in the initial release version of Studio 10.2 used HTML templates in the html/design folder to render the web view in the remote form in design mode. Note that design mode now uses the same template as runtime mode, either jsctempl.htm, or the \$htmltemplate from the design task, so jsctempl.htm no longer needs to be present in the html/design folder. The HTML file used for design mode is still generated in the html/design folder, but only to render the Web Preview of the form.

JS Themes

You can now edit the current or selected theme from the JavaScript Theme selector dialog (opened with Ctrl-J when editing a remote form) by Right-clicking on a theme or background of the dialog and selecting the Open JavaScript Theme Editor option; this opens the selected or current JS theme.

Enter & Esc Keys in Subforms

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

JavaScript Components

JS Edit Control

Incompatible input types are now prevented from being used with JS Edit Input Masks. For example, the kJSInputTypeNumber and kJSInputTypeEmail values of \$inputtypes are incompatible with JS Edit input masks. If \$inputtype is one of these values, and \$inputmask is set, the input element will use the text type (effectively kJSInputTypeDefault).

JS Button

When `$textishtml` for a JS Button control is set to `kTrue` the text in `$text` is treated as HTML. The HTML needs to be valid for it to be rendered, including when used as the contents of a `<p>` element, so for example you cannot use a `<p>` element inside another `<p>` element.

Field Styles for Complex Grids

The `$rowdividerlinestyle` is now assignable at runtime and by `$fieldstyle`. As `$rowdividerlinestyle` is a custom field in a `$fieldstyle` it gets assigned at runtime, and is treated like any other runtime property change, therefore it is now assignable at runtime. Note that `$rowdividerlinestyle` changes just the border between each row in a Complex grid, unless `$rowborder` is set to `kJSborderPlain`, in which case it also effects the border around the client, i.e. the section of the complex grid which contains the rows.

SVG Icons

More SVG icons have been added to the 'material' iconset, including icons for eating out (restaurant and café), travel (bus, train, car, bicycle), and so on. You can find more SVG icons on the Google Material icons website, and add them to the material iconset in Omnis (html/icons folder: note Omnis uses the 'black rounded' type). Alternatively, you can source other SVG icons and create your own new icon sets. You must convert any SVG files to 'themed SVG' icons using the SVG Themer tool (Add-ons>>Web Client Tools option) if you want to use the icons with themes in the JS Client.

PNG Icon Editor

You can now sort icon pages in ascending or descending alphabetical order in the Icon Editor (Tools>>Icon Editor option used for editing PNG icons) by clicking on the 'Pages' title above the list of icon pages.

Code Editor

Export List or Row Variables

You are now able to export the complete contents of a list or row variable from the Variable menu to a tab-separated file. There is a new menu item "Export Tab Separated..." that appears in the Variable menu for list and row variables, in the same location as the "Copy Value" option that appears for various simple data types. When selected, it prompts for the path name of a file that receives a tab-separated value representation of the list or row.

The output file is UTF-8 with a UTF-8 byte-order-marker. The first export row comprises tab-separated column names. Simple types in the list are exported as their actual value, whereas types such as lists are output as an information string, e.g. (5 Lines). If the characters tab, carriage return, linefeed or backslash occur in the data, they are exported as escapes: `\t`, `\r`, `\n` and `\\` respectively. If a column has a #NULL value, it is exported as the text NULL.

Code Folding

You can now remove code folding from all the methods in a class or all classes in a library. All classes that can contain methods now have the method `$removecodefolding` which removes code folding from all methods in the class, and returns the number of methods from which code folding was removed. For example, to remove code folding from all methods in all classes in a library, execute:

```
Do $libs.library.$classes.$sendall($ref.$removecodefolding())
```

In addition, the option 'exportcodefoldingstate' has been added to the \$exportimportjsonoptions Omnis Preference (\$root.\$prefs) to control whether or not the code-folding state in the methods in your library is exported; the option is set to false by default so the code folding state is not exported.

Libraries

JSON Import Option

There is a new boolean option 'importtreatsunknownpropertyaswarning' in the \$exportimportjsonoptions Omnis Preference (\$root.\$prefs) to treat unknown properties in imported JSON as a warning; it is true by default.

JSON Import Error Messages

Error messages have been improved when an import JSON fails due to the inability to parse a method line; the text that cannot be parsed is now included in the error message.

Omnis Environment

Help System

All HTML pages used to create F1 style Help systems using the Omnis Help Project Manager (available in the Tools menu) must now be UTF-8 encoded. Due to Character set issues building help word indexes, all HTML pages used with the Help Project Manager (and any additional text files such as the _exclude files) must now be UTF-8 encoded.

Window Components

OBrowser

The property \$donotredirectconsoleto tracelog has been added to OBrowser. If true (the default), browser console messages generated by OBrowser are not redirected to the Omnis trace log.

Functions

split()

There is a new split() function that allow you to split a string at the specified delimiter (comma is the default delimiter).

```
split(string[, delimiters=',', stripWhitespace=kFalse])
```

Splits the string at the character(s) in delimiters and returns a list of the resulting substrings. The function strips leading and trailing whitespace from each substring if stripWhitespace is kTrue (default is false). The function is available in both normal methods, and client-executed methods.

sys(192/292)

There is a new item "sys192ListRowLimit": N in the "defaults" section of config.json which allows lists (and rows) with up to N rows to be included as a third column in the output parameter data for the sys(192) and sys(292) functions (note: sys(192) returns the method stack as a list, and sys(292) returns the calling method).

If $N \leq 0$ (the default) then sys(192/292) behave as before. If $N > 0$, then each parameter in the parameter list stored in each line of the sys(192/292) list has a third column, which for lists and rows contains the actual list (or row) data, if the list or row has less than or equal to N lines. In all other cases (not a list or row, or line limit N exceeded) column 3 is empty.

What's New in Omnis Studio 10.2

For Omnis Studio 10.2 the appearance and useability of many of the JavaScript components has been greatly enhanced with the introduction of color *themes* and support for *SVG icons*. In design mode, *position assistance* is provided to help you arrange objects on a remote form, plus remote forms are now displayed in a *web preview* in design mode so you can see exactly how your forms will look at runtime.

This release also includes many enhancements in the Code Editor, including *Code Folding* and *Word Wrapping*, plus you can now edit your code when using the Remote Debugger. For the thick client, there is a new *Token Entry Field* and *Breadcrumb* control, plus Page panes can be displayed as Side panels improving the UX for desktop apps.

The following features have been added to Studio 10.2:

❑ JS Client Themes and Appearance

The appearance and useability of the JavaScript components has been greatly enhanced with the addition of *JS Themes* for managing colors used throughout your application; some of the JS controls now have *animations* and other *visual effects* to improve the UX for your apps; plus the *default size* of some of the components has been increased to better cater to touch devices

❑ SVG Icons

you can now use *SVG image files* for icons for JavaScript components and window controls; SVG images are vector based and are inherently scalable, therefore a single file can provide multiple icon sizes; specifically, an SVG image will scale to fit the icon area available in a control; and for the JS client only, *SVG icons* can be *themed* which means they change color to match the current theme

❑ Position Assistance

colored visual guides are now displayed automatically when you *move or resize objects* using the mouse (pointer) in a remote form, report or window class design screen; as you move or resize objects, colored lines are shown and objects will *snap into position* to help you arrange the objects in a form or report

❑ Remote Form Design

When you design a JavaScript Remote form it is now displayed in a *Web Preview* (using the built-in Chromium browser) so you can see exactly how a remote form will look and behave at runtime in the end user's browser, including the use of the current theme and any other visual effects

❑ New JS Split Button Control and other enhancements

the new JS Split Button provides a dropdown menu of choices on a single button; new style & positioning properties for the *Data Picker* for Edit controls and Data Grids; you can now send an SMS message to multiple recipients in the Device Control; plus the *\$inputmask* property has been added to JS Edit controls

❑ Method Editor & Code Editor

the Code Editor now supports *Code Folding* allowing you to collapse and expand code constructs, to improve readability and code manipulation, while *Word Wrapping* allows long lines of code to wrap onto the next line; there is a new *Search* box above the Method Names tree allowing you to find specific methods or filter the list; plus *built-in methods* for a class are now shown in the method list

- ❑ **Remote Debugger**

You can now edit methods and code while stepping through live code in the *Remote Debugger*; prior to this, code could only be viewed in read-only mode while using the remote debugger
- ❑ **MultiProcess Server**

The Linux Headless Server can now be run in *MultiProcess Server* (MPS) mode which can utilize the multi-core processors on your server, providing performance improvements for your server based, web and mobile apps
- ❑ **New Window controls**

the new *Token Entry Field* allows the end user to enter text which then becomes tokenized (a single block), similar to the recipient field in email programs; the new *Breadcrumb* control can be used to display the end user's "location" within the hierarchy of an application; and the *Check Box* control now allows a "horizontal" mode which behave like an "on/off" slider switch (all for thick client only)
- ❑ **Side Panels**

a Side panel is a vertical panel containing clickable options or other content that can be added to the left or right of a window, using a page pane, or scroll box; a side panel can be shown automatically or linked to a menu control to allow it to be opened or closed manually (for thick client only)
- ❑ **Toast Messages for desktop apps**

Toast messages are small notifications that that can be "popped" in your desktop application to alert the end user about something; this enhancement allows you to open toast messages in your desktop apps, via a window instance for example, using a new \$showtoast method
- ❑ **Drag and Drop for system files**

Support for dragging and dropping operating system files and file data (in the thick client) has been simplified providing more control over files and data in your event handling code
- ❑ **Regular Expressions**

the PCRE2 library has been added to Omnis to support regular expressions in your Omnis code or for Find and Replace; the PCRE2 library (*Perl Compatible Regular Expressions* version 2) is an open source library of functions that provides syntax and semantics like Perl 5 for defining a search
- ❑ **OAUTH2 Authorization and OW3 Workers**

there is a new OAUTH2 Worker Object providing general support for OAUTH2 authorization for the OW3 worker objects; the HTTP, IMAP, POP3, and SMTP workers have been modified to support OAUTH2 via the new OAUTH2 worker; plus there are some enhancements to the IMAP, HASH, and the FTP workers
- ❑ **OpenAPI for Web Services**

Omnis now generates an OpenAPI 3.0.0 definition for a RESTful web service as well as Swagger 2.0; OpenAPI is a more up to date version of the RESTful API description format, and Studio 10.2 now generates OpenAPI 3.0.0 definitions, as well as Swagger 2.0 definitions
- ❑ **Localization for JS Client**

localization for the JS Client has been optimized, reducing data size for applications that support multiple languages by only loading language file(s) as required; plus German, French, Italian and Spanish are supported by default, while support for other languages can be added
- ❑ **Omnis Datafile Migration**

The DML emulator has been substantially re-written for Studio 10.2 to improve performance; this allows you to convert an Omnis data file to either SQLite or PostgreSQL

JavaScript Components

The following new features and enhancements are for JavaScript components.

JS Client Themes and Appearance

The appearance and useability of the JavaScript components has been greatly enhanced to help you design better and more consistent UIs, as well as improve the accessibility for your applications created using the JS client. The enhancements include:

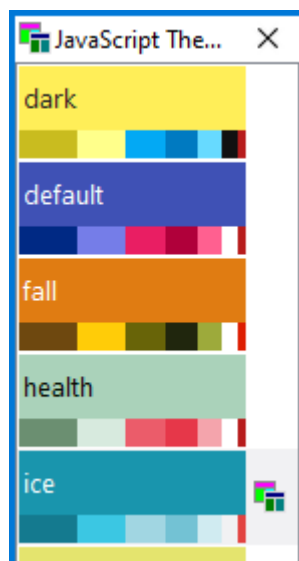
- ❑ the introduction of color **Themes**, to help you apply colors consistently across your web and mobile applications
- ❑ support for **SVG icons** which can be scaled to any size, and colored or styled using the theme in your app
- ❑ the addition of **animations** and other visual effects to enhance the UX in your apps, such as a ripple effect for button clicks, plus improved border highlight and shadow effects to show the focus
- ❑ plus the default **size** of some of the components has been increased to cater to the *touch* interface on tablets and phones

The appearance enhancements have been guided in part by Google's 'Material' design system, including the use of *primary* and *secondary* colors, to modernize and improve the UI for your JS client applications.

There is a new example app under the **Samples** option in the **Hub** in the Studio Browser called 'JS Input Border and Button Styles' to highlight some of the appearance changes for Edit controls and Buttons.

JS Themes

You can now apply a consistent set of colors to components on a JavaScript remote form by selecting colors defined in a *theme* – underlying a theme is a set of CSS styles which are applied to controls at runtime in the browser. Omnis has a number themes which you can use to style your JS client applications: a **default** theme, which provides an effective and pleasing UI across all JS controls and devices, and a range of different color themes, such as the **dark** theme, which provides an alternative set of darker colors.



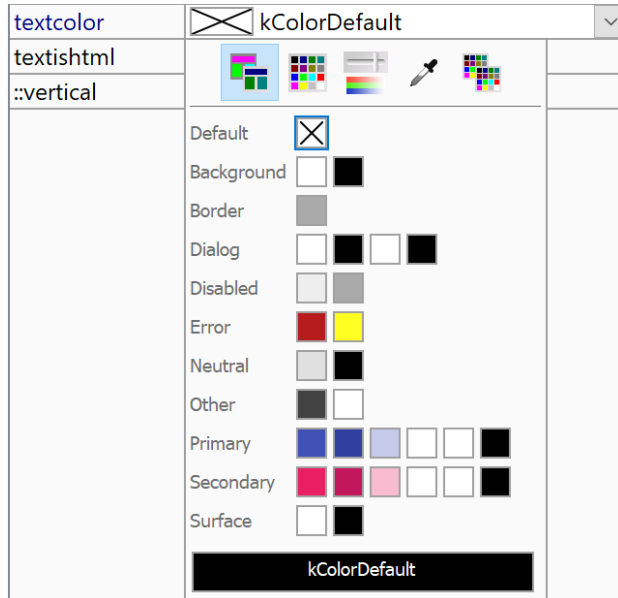
When designing a remote form, you can change the current theme in the JS Theme Select dialog by pressing Ctrl-J, or select JavaScript Theme from the **View** menu. To select a theme, click on the theme preview and close the dialog. The selected theme is applied to the current remote form and to all the remote forms in your library since the theme is an Omnis-wide preference.

The current theme is stored in a new Omnis root preference, **\$javascripttheme** (in \$root.\$prefs), which is set to the *default* theme initially, and controls which theme is

used to render themed colors for all remote forms in design mode (but you can set or change the theme on the client using the 'settheme' client command; see later).

Selecting Colors

When you select the color for a JS control in design mode in the Property Manager, you can now choose a theme color from the color picker, under the new Theme color button in the color picker toolbar (existing users should note that the color brightness button & setting has been removed). For example, select a button, click on the Text tab in the Property Manager and click on the color picker for \$textcolor.



The color setting for most properties, such as \$textcolor, is set to **kColorDefault**, which means the appropriate color from the current theme is used. If a text color property is set to kColorDefault, and it sits on an element with a background color which comes from a themed color constant, the text will be rendered in the associated <theme color>Text color. For example, if a button's \$buttoncolor is set to kJSThemeColorPrimary and its \$textcolor is set to kColorDefault, the text will be rendered using kJSThemeColorPrimaryText.

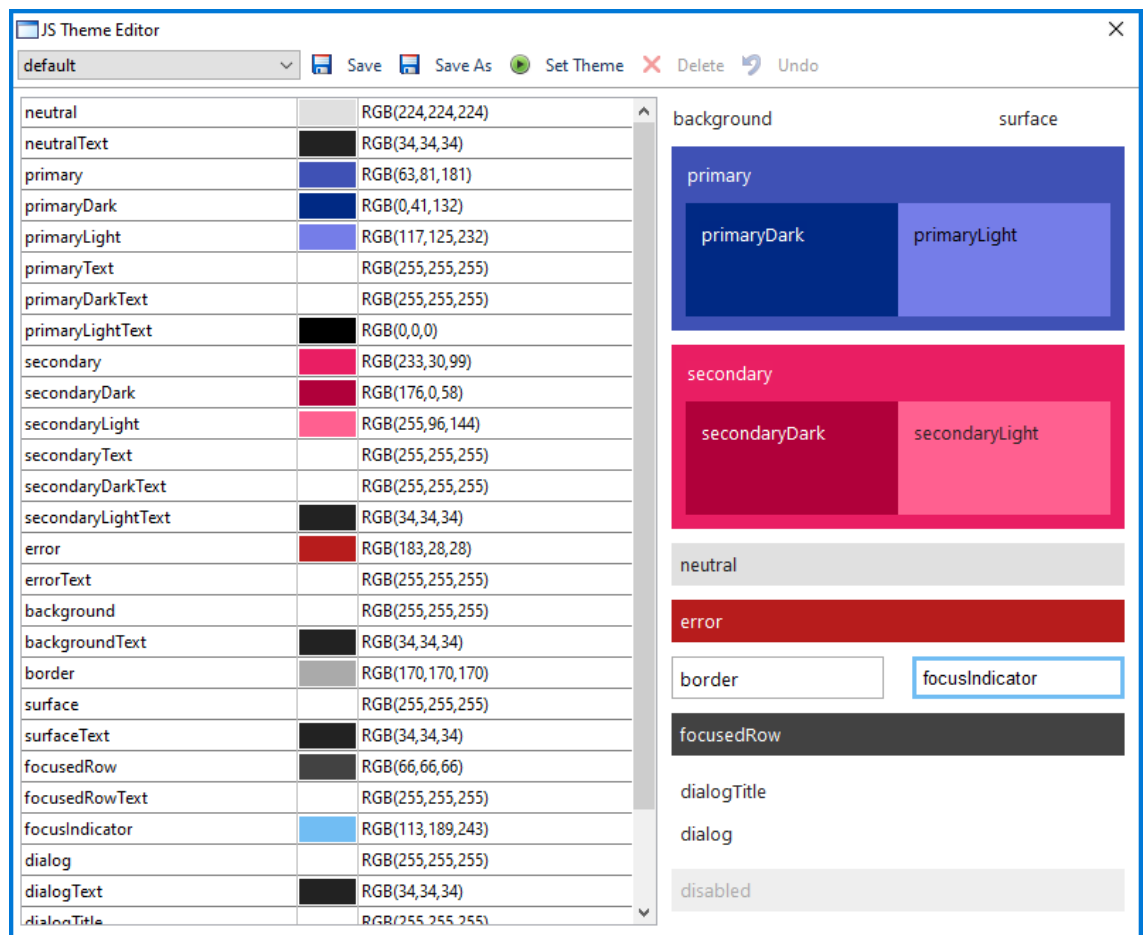
The colors defined in a theme and shown on the color picker have corresponding color constants, whose names begin **kJSThemeColor**, as follows:

kJSThemeColorBackground	kJSThemeColorPrimary
kJSThemeColorBackgroundText	kJSThemeColorPrimaryDark
kJSThemeColorBorder	kJSThemeColorPrimaryDarkText
kJSThemeColorDialog	kJSThemeColorPrimaryLight
kJSThemeColorDialogText	kJSThemeColorPrimaryLightText
kJSThemeColorDialogTitle	kJSThemeColorPrimaryText
kJSThemeColorDialogTitleText	kJSThemeColorSecondary
kJSThemeColorDisabled	kJSThemeColorSecondaryDark
kJSThemeColorDisabledText	kJSThemeColorSecondaryDarkText
kJSThemeColorError	kJSThemeColorSecondaryLight
kJSThemeColorErrorText	kJSThemeColorSecondaryLightText
kJSThemeColorFocusedRow	kJSThemeColorSecondaryText

kJSThemeColorFocusedRowText	kJSThemeColorSurface
kJSThemeColorNeutral	kJSThemeColorSurfaceText
kJSThemeColorNeutralText	

Theme Editor

You can create new themes, or modify an existing theme using the JS Theme Editor, available under the **Add-Ons > Web Client Tools** menu option and select **JS Theme Editor**.



The editor provides a preview of the current theme on the right side of the editor screen, and you can click on an area or text item within the preview to view or set its color (you can also set colors by clicking in the list on the left).

The colors in a theme are categorized as **Primary** and **Secondary**, plus there are specific color for **errors, borders, dialogs**, and so on. The primary colors are used throughout your application and set the general tone or style of the theme, while the secondary colors provide an accent to certain parts of the UI.

Creating a new theme

To create a new theme, *you can duplicate an existing theme* and make any changes to the copy. To do this, open the Theme Editor, select a theme from the dropdown list or use the *default* theme (selected initially by default), click on **Save as** and give the new theme a name – then change individual colors and use the **Save** option to save any modifications. The **Set theme** option sets the \$javascripttheme preference to the theme currently shown in the editor. If you make any modifications to the current theme, all open remote forms will be updated automatically.

A theme is stored as a **.json** file and an associated **.css** file in the 'html/themes' folder. When deploying your application, the themes folder and its contents must be copied to the corresponding location on the Omnis App Server.

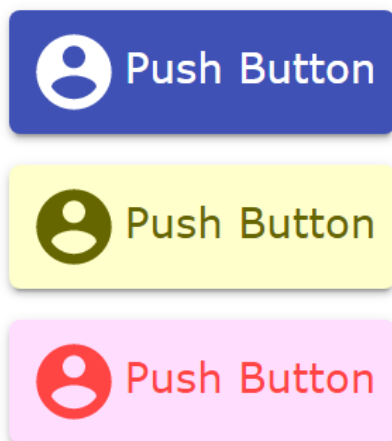
When designing the colors in a new theme, you may want to follow the guidance provided by the Google Material design system, which may help you create a theme containing colors which complement one another and provide maximum usability and accessibility across different platforms and devices. Google provides a [Material Color Tool](#) which you may find useful to create a set of complementary colors for the dark/light variants.

Themed Icons

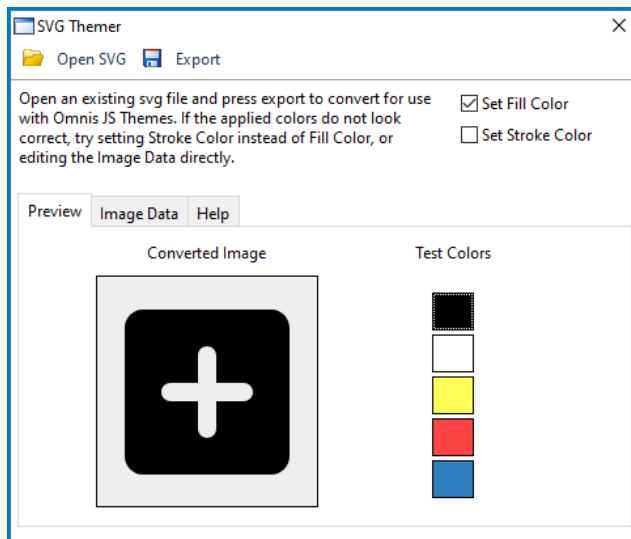
This version of Omnis supports the use of SVG images for component icons (see the next section in this doc regarding how to use SVG icons). For the JS client only, SVG icons can be "themed" which means an icon will be tinted using the control's text color as specified in the current JS theme (the 'fill' color in a themed SVG file is set to the text color from the theme). This allows *a single themed SVG icon file* to be used with different themes and its color is set automatically.

Omnis includes an icon set named 'material' which contains over a 100 themed SVG icons (note this icon set can only be used with the JS Client, not window classes since they do not support themed icons). The material icon set is located in the 'html/icons' folder and if you have used any of the icons in your app the icon set needs to be copied to the Omnis App Server when deploying your application.

The following are examples of a single icon from the material icon set with different color themes applied (note the icon is rendered using the button text color):



SVG icon files can be 'themed' using the **SVG Themer** tool under the **Add Ons > Web Client Tools** menu option. You can open a single SVG file, preview it using one of the test colors (the preview colors are not saved to the file), and save it using the **Export** button.



The SVG Themer tool converts a standard SVG image file into an Omnis themed SVG file format: specifically, the first element in the root svg element in the original file is converted to a 'g element' with fill="var(--om-tint-color)" and id 'omTheme' which reference the color from the current theme. The Image Data tab shows the source for the converted SVG file which you can edit if required, although the converter converts the the SVG file as necessary.

(Beta testers should note that the format of themed SVGs has changed since the beta4 of Studio 10.2, so you will need to convert any SVG icons again using the release version.)

Like other SVG icon files, any themed SVG icons need to be placed in an icon set folder. For example, you could create or acquire a set of SVG icons and convert them using the SVG Themer tool ready for use in your JS client apps.

HTML Template & JS Client theme setting

The JS client's theme is set in the 'data-themename' attribute in the omnisObject div in the HTML file for your remote form, e.g. data-themename="dark".

The special value of "_JT_" is used in the HTML template (jsctempl.htm) which is replaced with the current value of \$javascripttheme when Omnis generates the HTML file for your remote form.

In addition, the 'data-appid' attribute specifies the application a page belongs to. It defaults to '<lib name>.<form name>' each time a form is tested (the '_APPID_' placeholder in the template .htm file is replaced when a form is tested).

Changing the Theme

You can change the theme on the JS client in your code using a new 'settheme' client command (\$clientcommand) which takes a row parameter whose first column is the name of the new theme. Note that a remote form needs to be reloaded in the browser for a change of theme to take effect. Once you have set the theme using 'settheme', the client stores it in the client localStorage and will use that theme for subsequent visits to the page. To revert back to the default theme specified in the HTML page, you need to call the 'settheme' clientcommand, passing an empty string as the theme name (or clear the client's localStorage).

The current theme: \$construct

The current theme is passed in the \$construct row parameter, in a column named *theme*.

Note for existing users: active color properties

The JS client now uses a ripple effect for which the colors are generated automatically, so the following properties are no longer relevant and have been removed from the Property Manager (they will continue to work in existing apps):

- Nav bar - `$buttonpressedcolor`
- Split button - `$activebackcolor`
- Toolbar - `$toolbaractivecolor`

SVG Icons

You can now use SVG images for icons for JavaScript Remote Form components and in most other places that currently support bitmap images for icons, as in previous versions. Specifically, you can use SVG image files in an Icon set, alongside any existing icon sets containing PNG files, and these will appear in the Select Icon dialog when you need to assign an icon to a JS component. (You can also use SVG icons for Window class controls, but they cannot be themed, see below.)

SVG images are vector based and are inherently scalable, therefore a single SVG file can provide multiple sizes for icons – a single icon file will scale to fit the icon area available in a control (unless you fix its size, see below). By contrast, component icons in previous versions only supported PNG graphics and therefore you had to create a separate image file for each icon size or resolution you wished to support and place all the separate files in an icon set in the Omnis tree. In addition, a single vector-based image will have a much smaller file size than multiple PNG files, giving your app a smaller footprint on the client.

Platform support

On macOS, SVG icons only render in the thick client when using macOS 10.13 or later. On Windows, SVG icons only render when using the Windows 10 Creators Update or later. In general, support for SVG in Windows is more limited than on macOS, for example, Windows does not support classes in SVG files – read here about Windows SVG support:

<https://docs.microsoft.com/en-us/windows/win32/direct2d/svg-support>

Creating SVG Icons

You can create your own SVG icons, or you may be able to acquire a set of icons from a third-party, either paid-for or for free (subject to the appropriate licensing), such as the icons provided in the Material design scheme from Google and issued under the Apache License Version 2.0 (<https://material.io/resources/icons>). We have selected over 100 of the Material icons (from the black, rounded style) and placed them in an icon set folder called 'material' under the main 'html\icons' folder, and you are free to use these in your Omnis applications (with the proper attribution in your product licensing); note these Material icons have been 'themed' and therefore support the new JS Themes. You can download other icons from the Material website and add them to this folder, if required.

SVG image files must be saved with the .svg file extension (see naming below) and should be placed in a subfolder in the 'iconsets' folder in the Omnis tree; the name of the sub-folder becomes the name of the icon set, and in order to use the icons, the icon set name needs to be added to the list of icon sets in the `$iconsets` preference in your library (note `$iconsets` can now take a list of icon set names).

From our testing, we found that Adobe® Illustrator® allows you to export vector images in SVG format, and on the export to SVG options dialog you can select the 'Inline Style' option to ensure classes are not used in the output SVG. There are many other image editors that can output SVG.

Themed Icons

In order to work with the new JS Themes, an SVG icon needs to be converted to a Themed SVG file. Themed icons only appear in the Select Icon dialog for JavaScript remote forms and Remote Menus (not for window classes or controls, since they do not support themes). The Material icons in the 'material' icon set have been themed.

Using SVG Icons

If a JavaScript component can support SVG icons, and most do, then the icon IDs (names) of any SVG icons will appear in the Select Icon dialog when you assign the icon via the Property Manager and the Select Icon dialog (if a component does not support SVG icons, then they are not shown in the Select Icon dialog).

In general, SVG icons are supported by any controls that previously required an icon, including the following classes or features:

- Remote Form class components (JavaScript Client controls), including buttons, menus, toolbars, lists, tabs, check boxes
- Window class controls (thick client), including menus and toolbars, together with some external component window controls including clock, treectrl, html icon link, hypertext, etc.
- Styled text, including styled text on reports sent to the Omnis PDF report destination
- The background icon for the main Omnis window on the Windows platform (\$root.\$prefs.\$backgroundiconid)
- The \$componenticon class property

You should note the following for JS controls only:

- Some JS controls use background-image CSS, so when using an SVG image, it will not always scale as expected if the aspect ratio in the SVG is fixed, and the desired dimensions of the background-image do not have the same aspect ratio.
- JS Popup menu and JS Navmenu controls have hot iconid properties – in this case, the hot and equivalent non-hot iconid properties must either both use SVG or both use PNG

Naming and Icon Sets

The base icon ID of an SVG icon is the name of the SVG file, without the file extension, and converted to lower case, up to a maximum of 32 characters. The naming restrictions for SVG icons are as follows:

- The base icon ID *must not represent an integer* (the icon ID had to be an integer for PNGs, but does not have to be for SVG image files)
- The base icon ID *must not contain the characters + # , ; = ?* (plus, hash, comma, semicolon, equals, or question mark); note + is used to add a size restriction, see below

An icon ID or name can now be either an integer or a string, and integer icon IDs work exactly as they did before (the naming of PNG icon images remains the same).

You cannot use the same file name with different case in an icon set folder, plus it's always good practice to make icon IDs or names unique across different icon sets, since the icon with the first instance of a specific icon ID or name is used.

Any errors related to the naming requirements are written to the icon set log file, which is in the folder logs/iconsets, in the data part of the Omnis tree.

Multi-state Icons

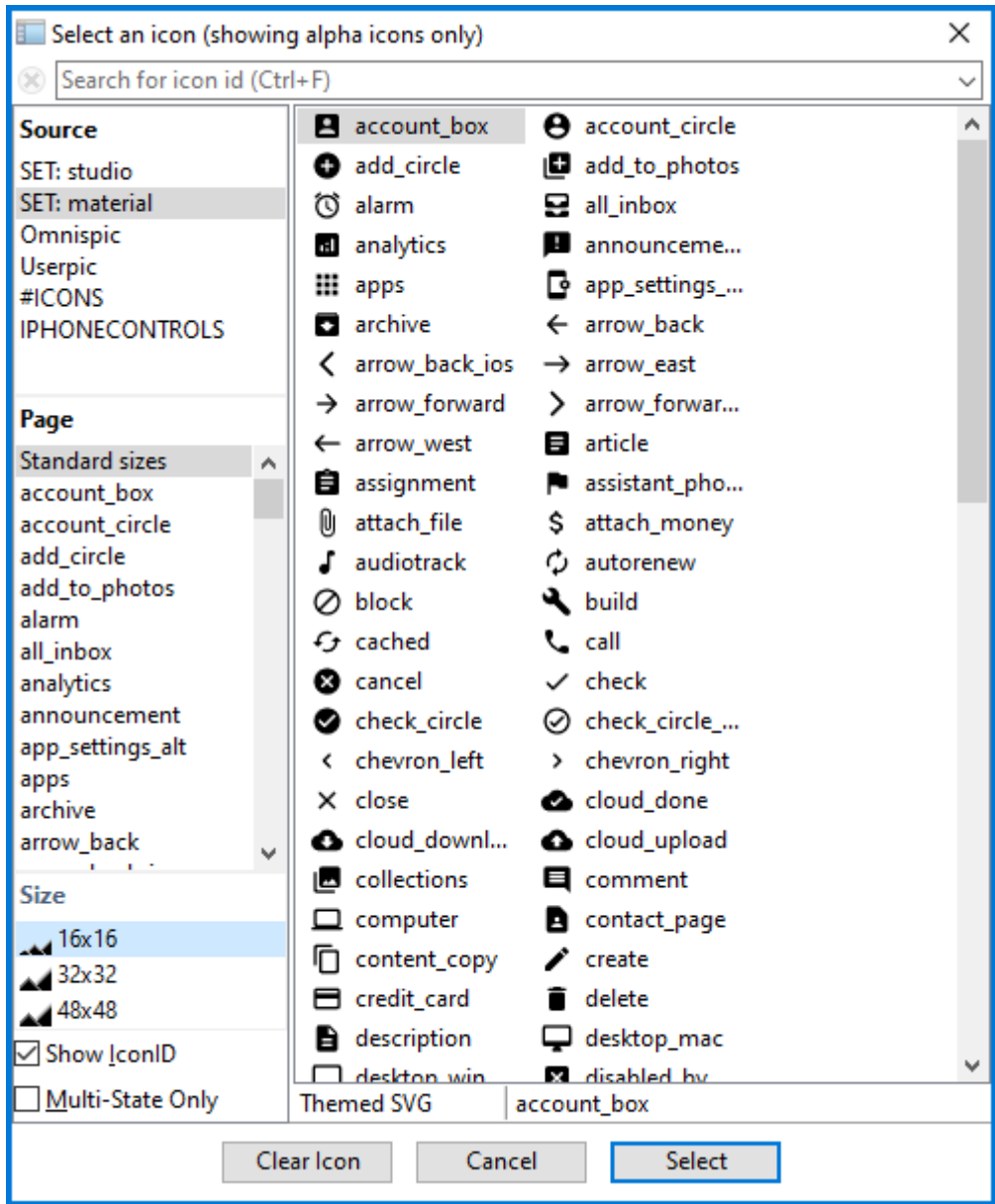
If you want to include icons for different states of a control (for example, checked, highlighted, and checked highlighted for a check box control), you can include separate SVG files with a suffix in their name:

- `_c` for checked
- `_h` for highlighted
- `_ch` for checked and highlighted

For example, SVG files for a check box could include the files: `checkbox.svg` (for the unchecked icon), `checkbox_c.svg`, `checkbox_h.svg` and `checkbox_ch.svg` (for the different states). These 4 files all result in a single icon with id 'checkbox', and Omnis will select the correct SVG file according to the state of the checkbox.

Selecting an SVG icon

The Select Icon dialog has had a few modifications to support SVG icons; the screen below shows the 'material' icon set. When you select an icon set containing SVG icons, the page list in the dialog shows an entry for standard sizes, and a full-page entry for each individual SVG icon. The standard sizes include all the SVG icons in the set, since SVG images will scale to any size. When you select a full page SVG icon, the first line of the size list shows the default size read from the SVG file (converted to Omnis design pixels), with the text (kDefSize) appended to it.



The select icon dialog has a new status bar area that shows the type of icon (PNG, SVG, Themed SVG or Icon page entry), and the ID (name) of the icon.

There is a Search box on the Select Icon dialog that allows you to search for an icon or filter the icons shown using the icon ID or name.

Fixed and Custom Icon Sizes

An SVG icon will always expand to fit the available space within a control, but it is possible to fix or restrict the size of an icon by adding size information to the end of the icon ID name. The size information has the syntax `+<w>x<h>` where `<w>` is the integer width and `<h>` is the integer height. For example, an SVG icon ID could be any of the following:

- testsvg (unrestricted size)
- testsvg+16x16 (restricted to 16x16, for example, for a menu)
- testsvg+32x48 (restricted to 32 wide x 48 high)

When selecting an SVG icon, the size list includes the configured sizes from config.json, and the current size of the icon, in addition to the standard sizes and kDefSize. There is a + button in the heading of the size list that allows you add a new size. There is an option on the dialog to add the new size to config.json.

There is a new configuration item called 'customSizes' in the 'svg' section of config.json that allows you to add other sizes. The size list in the Select Icon dialog will show any other sizes specified in the config.json file:

```
"svg": {
  "customSizes": [
    "256x256",
    "64x64",
    "128x128"
  ]
}
```

When a custom size is selected in the size list for a full page SVG icon, in addition to the + button, there is a - button which you can use to remove the size from the list, and optionally remove it from config.json.

Omnis uses the default width and height specified in an SVG file to determine the aspect ratio of the icon image. To obtain this, Omnis looks for the width and height attributes of the svg element in the SVG file and uses these if present. If width and height are not present, Omnis uses the viewBox attribute of the svg element to determine the aspect ratio. In this case, you can add a size using the + button in the Select Icon dialog, and use the Keep Aspect Ratio option, to fix the aspect ratio.

Icons for Lists

Certain controls, such as the Icon Array, use a list column to contain an icon ID. To make use of SVG icons, this column now needs to be defined as Character. Where you use a mixture of SVG icons and existing icons, the icon IDs can be specified as strings or integers as appropriate.

Icon Caching

Prior to this version, Omnis cached every icon set icon in memory, as bitmaps. To handle SVG support, you can now control the cache size for all icon sets (using PNG and SVG icon image files). There is a new entry in the 'defaults' section of config.json called maxCachedIconSetBitmaps. This is an integer, which defaults to 1000 bitmaps. If Omnis needs to create a new bitmap for an icon from an icon set, and the current number of cached bitmaps is at this limit, Omnis will free up the least recently used bitmap.

Multiple Icon sets

In addition to support for SVG icons, you can now specify multiple icon sets for a library. Therefore, the \$iconset library preference has been renamed to \$iconsets, and can now accept a comma-separated list of icon set folder names, or a single icon set name as before. The icon set folders are searched in the order specified in the property, followed by the Studio icon set, then the library #ICONS system class, and finally the icon data files Omnispic and Userpic.

Icon Search order

The Select Icon dialog now shows icon sets in the order in which they will be searched when an icon is referenced. If there is a duplicate icon name, then a component or window control will display the first icon found by the search. The Select Icon dialog will show the icon from each icon set even if the icon will be overridden by the search order.

During SCAF generation, for the serverless client, the Omnis Server now passes all the files for all icon sets in \$iconsets to the serverless client library.

Multi-state Icons

The Select Icon dialog will now only display multi-state icons for controls that require a multi-state icon, such as check boxes. In addition, there is a check box on the Select Icon dialog so you can display the multi-state icons only.

JSON Export-Import

The new icon ID syntax is handled when exporting a class to JSON, and importing JSON to a class.

There are new flags specified in property tables to identify icon ID properties that support SVG icons. For the thick client, the flag is PROP_SVG, and for external components, the flag is EXTD_EFLAG_SVG.

Icon APIs

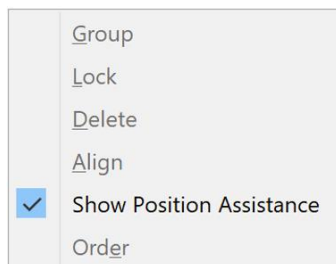
The bitmap APIs for both the core and external components now have overloads that accept a fldval or EXTfldval respectively, to represent an icon ID that can be either an integer or a string.

Position Assistance

Position Assistance provides visual guides (colored arrows and dashed lines) that enable you to easily align and distribute controls and other objects in a design window, that is, when you *move or resize objects* using the mouse or trackpad. The new position assistance is available when positioning objects in a JS Remote form, a window class, or in the report editor.

As you move or resize objects on a remote form (or report or window), colored lines are shown automatically, and objects will snap into position to help you arrange the objects in a form. Position Assistance is also provided when you use the Arrow keys to position or resize objects.

The context menu for the remote form (or report or window) has a new entry after the Align hierarchical menu, "Show Position Assistance", which toggles the new Position Assistance (default is enabled). There is a single setting for this, shared by all editors, that is saved to omnis.cfg when Omnis shuts down.



Position Assistance for *sizing* does not apply when Size to Grid is turned on, and for *moving*, it does not apply when Align to Grid is turned on.

The positioning lines are drawn using the **colorhighlight** color in the system group of appearance.json. There is a new entry positionAssistantKeyboardTimer, in the ide section of the config.json, that can be used to adjust the time that the position assistance remains visible after you stop pressing an arrow key; this defaults to 750 milliseconds.

Positioning & Aligning Objects

When the Position Assistance is enabled, Omnis gives precedence to distribution over alignment, and within alignment it prioritises the top edge, over the center, and the center over the right edge. As soon as a visual guide is displayed for a target, any other targets that would also cause the object to move in the same axis are dropped.

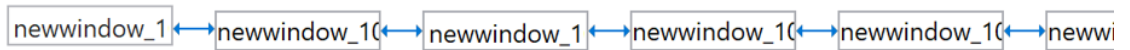
As you move or size objects Omnis displays a visual guide when the object(s) being moved or sized are within +/-2 pixels of a specific alignment or distribution target, e.g. an alignment target is the top edge of another object or objects. When you release the mouse, the objects snap to the displayed target. Position Assistance is applied to objects dragged from the Component Store, as well as objects being moved or sized

within a design window. Position Assistance is provided when moving an object even if the adjacent objects are contained inside a container field.

When sizing objects, assistance is not provided if the objects being sized have more than a single container, that is, the component that is the parent of the objects – this can be more granular than a field, such as for complex grids, there are several containers such as the row and header sections.

Distribution

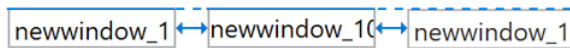
Position Assistance attempts to *distribute objects* by allowing them to be evenly spaced. The visual guide for distribution is a line drawn between the objects with arrow heads.



The guides are drawn for as many objects as possible, immediately adjacent to the object(s) being moved or sized. Position Assistance works best when objects are already reasonably well arranged, either vertically or horizontally, so for more complex arrangements, with overlapping fields may result in no visual guides being presented.

Alignment

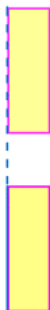
Position Assistance attempts to *align objects* by giving them the same *top* or *bottom* coordinate, or *centered* relative to each other. When you try to center objects, you only get visual guides when moving objects, and when the appropriate side of the rectangle representing the objects being moved either fully encloses or is fully enclosed by the appropriate side of the object in which it is being centered. The following illustrate how the Position Assistance is applied for different cases when aligning objects.



Top alignment



Bottom alignment



Left alignment



Right alignment

Positioning for Paged Panes (Container fields)

Assistance is provided to help you align fields inside a container field, such as a Paged Pane. In addition to the left/right, top/bottom positioning, when you move an object inside and *near to the center of a container*, a line across either the vertical or horizontal center of the container is drawn and the object will snap to the line.



When positioning objects *inside a Paged Pane* (or any container), Position Assistance is only provided for the controls *within the Paged Pane* itself, so objects outside the Paged Pane are not included in the current object grouping. Similarly, if you are positioning objects *outside, but near to a Paged Pane*, the objects inside the Paged Pane are not included in the current grouping.

Positioning for Complex Grids

Position Assistance is provided within each section of a Complex Grid, that is, the row and header sections of a Complex Grid, and the above behavior for container fields applies to each section independently.

Position Assistance for Reports

Position Assistance is available for fields and controls when designing a report, however the behavior is different for *non-floating report fields*. When moving or sizing objects in the report editor, Position Assistance is not provided for the vertical axis if any of the objects being moved or sized are not floating.

Remote Form Design

When you create or modify a JavaScript Remote form class the form design window is now displayed in a *Web Preview* (using the Chromium built-in web browser), so you can see exactly how your form will look at runtime in the end user's web browser. Specifically, JavaScript controls (and JSON-defined controls) will now look the same in design mode as they will do at runtime in a web browser, including the visual effect of any CSS styles you have applied to the controls (using `$cssclassname`). In addition, your remote form and its controls will be displayed using the current JavaScript theme. There is a new folder in the 'html' folder named 'design', in which the HTML for remote form design mode is generated; note this folder is only for design mode and is not required when you deploy your application.

Note to Existing users: Using old design mode

Existing users should note that there are a few differences between the new Web Preview mode for remote forms and the old design mode, as follows:

- There is no *design grid* available in the new Web preview mode, so `$showgrid` is not present (`$showgrid` is not available if you switch to the old design mode).
- Rulers* are not supported in the new Web preview mode, so the remote form context menu does not have an option to show Rulers.
- Design DPI scaling* does not apply in the new Web preview mode.
- The JS client now uses box-sizing `border-box`, so the appearance of control borders may be different.

- ❑ It is possible an exception will occur in the JS client running in the new Web preview mode: this does not have any effect on the validity of the remote form class. If this occurs, a message will be displayed for 5 seconds, and the error will also be logged to the trace log. In this case, you should close and re-open the editor after an exception.

You can revert to the old design mode for remote forms, either by holding the Shift key down when opening a remote form in design mode, or by changing the config.json entry 'useObrowserForRemoteFormDesign' in the ide section.

HTML control

The HTML control has a new property, \$showruntimepreview, which defaults to kTrue, which ensures the HTML is rendered in the remote form rather than showing the HTML code text. If \$showruntimepreview is false, the HTML code text is shown, but it cannot be scrolled inside the control in the design window.

JS Split Button Control

The Split Button control is a new JavaScript component: it combines a standard button with a dropdown menu, allowing you to provide multiple, alternate actions grouped together in a single button control. The Split Button is like the Send button in gmail as it provides two options in one control: a default *Send* option on the button and a *Schedule send* option via the menu.

The component is available for JavaScript remote forms as well as window classes, but there are some additional properties for the JavaScript control; note the split button for window classes is an external component which must be loaded via the External Components option on the Component Store.

The menu for the control is specified in the \$menuname property and must be a Remote Menu class for the JavaScript component, or a menu class for the Window class control.

The following example Split button has a Print option and a printer icon on the main button part, and it has options for printing to a Preview, PDF or File specified in a Remote menu class specified in the \$menuname property of the button control. In this case, a single click on the button would activate the Print to printer option, while clicking on the down arrow provides the other options (the images uses the 'professional' JS Theme).



Properties

The following properties are available for both the JS Remote form and Window class controls.

Property	Description
\$hotbackcolor	The background color of the control when hovered
\$activebackcolor	The background color of the control while pressed; active color is generated automatically if \$activebackcolor is kColorDefault
\$buttonborderradius	The radius in pixels of the corners
\$borderwidth	The width (0-7) of the edges drawn as the border of the control
\$arrowside	The position of the dropdown button on the control
\$textbeforeicon	If true, and the control has both text and an icon, the text is drawn before the icon
\$vertical	If true, the text and icon are arranged vertically
\$menuname	The name of the menu class, a Remote Menu class for the JS control, or a Menu class for the Window control

The following properties are available for the JavaScript control only.

Property	Description
\$menubackcolor	The background color of menu lines
\$menuhotbackcolor	The background color of menu lines when hovered
\$menutextcolor	The text color of menu lines
\$menuhottextcolor	The text color of menu lines when hovered
\$menudisabledtextcolor	The text color of disabled menu lines

Events

An evClick event is triggered when the main button area is pressed. In addition, for the JavaScript client only, the evOpenContextMenu and evExecuteContextMenu events are generated when the menu is pressed and in this case the pControlMenu event parameter is kTrue (when a Context menu is opened pControlMenu will be kFalse).

JS Edit Control

Input Masks

Support for input masks has been added to JS Edit controls with the addition of the \$inputmask property: this allows you to use a custom *input mask string* to control user input on a character level in data entry fields in a remote form. If the user enters an invalid character, the control will briefly become highlighted and the input will be rejected. For edit fields of character type, the data variable will contain mask characters. For number/integer fields, the data is the unmasked number value.

As a consequence of adding \$inputmask, there is a new system class #JSMASKS that stores the input masks for JS Edit controls for the library, and a new \$javascriptinputmasks notation group.

There are a number of differences between the existing Masked entry field on the thick client and the new input masks for JS Edit controls:

- On the thick client, the user must complete the masked entry field before focus can leave the field. This is not the case with JS masked edit fields - fields can be left partially filled.
- JS input masks do not support any of the 'control characters' which can be used on the thick client.

- ❑ The JS edit control does not have a `$formatstring` property (like the thick client masked entry field).
- ❑ The JS edit control has two unique properties: `$inputmaskguide` and `$maskvaluevalid`.
- ❑ JS input masks can be changed dynamically as the user types using the `$processmask` client method.
- ❑ There is visual feedback when entering invalid characters in a masked JS edit field.

\$inputmask

The value of `$inputmask` may contain a combination of fixed and special characters. Note that underscores cannot be used as these are used as placeholders.

Special character	Description
#	Any digit
@	Any character
a	Any letter
A	Any uppercase letter
n	Alphanumeric
N	Alphanumeric, uppercase
"ABC"	Any character from list
"A-D"	Any character from A to D inclusive
\	(back slash) Escape character (next character is displayed literally, use to escape special mask characters, double quotes or backslash)

\$inputmaskguide

The `$inputmaskguide` boolean property controls whether or not a guide is shown. If true, placeholder and non-placeholder mask characters are always displayed. If false, placeholder characters are hidden, and mask characters are only shown when the user reaches them as they type. The property is false by default.

\$maskvaluevalid

The `$maskvaluevalid` property is a boolean, read only, runtime only property. A value of `kTrue` indicates that the field is completed, and therefore valid.

\$processmask

A client method named `$processmask` can optionally be added to an edit control. This allows the mask to be changed as the user types. The method is called any time the value in the field changes, and receives a parameter `pInput` which contains the user input. Note that the user input parameter could contain anything as the event is sent before mask validation occurs (the mask needs to be updated before it can validate input). As a general use case, `$processmask` could be used to create the effect of optional characters.

Horizontal padding

When a library is converted to Studio 10.2, the `$horzpadding` property for all JS Edit controls will be set to 4 automatically if they were previously set to 0, which is the default for all new Edit controls; if `$horzpadding` is set to any other value it is not changed. After conversion, you can change the value of `$horzpadding`.

Vertical padding

The `$vertpadding` property has been added to the JS Edit control which allows you to add vertical padding to the text inside the control's border. The new property only applies when `$issingleline=kFalse` as single line edit controls are vertically centered.

JS Date Picker

Mode & Popup Style Properties

There are two new properties controlling the style of the Date Picker control. The `$datepickermode` and `$datepickerpopupstyle` properties control the mode (style) and popup style (positioning) of the date picker displayed when a date is entered into an Edit field (also applies to data grid cells and columns).

❑ `$datepickermode`

controls the type of picker to be displayed, one of the following constants:

`kJSDatePickerModeAuto`: Date picker type is assigned automatically based on `$dateformat`

`kJSDatePickerModeCalendar`: calendar type is displayed

`kJSDatePickerModePicker`: a picker type is displayed

❑ `$datepickerpopupstyle`

controls how the popup will be displayed, one of the following constants:

`kJSDatePickerPopupStyleAuto`: Popup style will be determined by device type

`kJSDatePickerPopupStyleInline`: Popup style will always be displayed adjacent to the control

`kJSDatePickerPopupStyleModal`: Popup style will always be displayed modal

(Note that Internet Explorer does not correctly display the modal type, and so falls back to inline on these clients.)

The inline style picker will position itself underneath the parent control, but from the right so it is closer to the icon which opens it. If there is not enough space beneath the parent control, the picker will be placed above, where space permits.

In addition, **Data Grids** have the new `$datepickermode` and `$datepickerpopupstyle` properties, as well as `$columndatepickermode` and `$columndatepickerpopupstyle`. The latter two work in the same way, but on the given column when `$userdefined = true`.

JS Data Grid

Tabbing through cells

The property `$tabthroughcells` has been added to JS data grids to change the action of the tab key while the focus is on the grid; it is set to `kFalse` by default. If set to `kTrue`, tabbing from a cell which is not being edited selects the next cell, or `Shift+tab` selects the previous cell. In addition, setting `$hcell` or `$vcell` now triggers edit mode if `$autoedit=kTrue`.

Column header height

The JS Data grid has a new property, `$columnheaderheight`, that specifies the height in pixels of the column header area. If set to 0 (the default) the header height will be the same as `$rowheight`.

Column header line breaks

You can now create multi-line column headers in a JS Data grid using a line break. You can use `\n` in the text for `$columnnames` to create a line break.

evCellValueChanged & pHorzCell

The `pHorzCell` event parameter of `evCellValueChanged` now references the column of the grid control itself, rather than the column of the data list belonging to the data grid as in previous versions. This has consequences for grids in which `$columndatacol` is used to map columns and you may need to change your code accordingly.

pDataColumnName

A new event parameter `pDataColumnName` has been added to the JS Data Grid events `evClick`, `evDoubleClick`, `evCellChanged`, and `evCellValueChanged`. The new parameter contains the data list column name (or number) when the event is triggered.

This is useful when columns in the data list do not map directly to the columns of the form data grid (that is, if `$columndatacol` is used to set the column order).

If the list column does not have a name, the parameter contains 'C1', 'C2', etc, so it can be used notationally. The value of the cell can be obtained with:
`iDataList.[pVertCell].[pDataColumnName]`.

Open Filter Method

The data grid has a new client-executed method `$openfilter` which can be called from `$init` to allow you to open the filter area in the grid when the form is opened.

❑ `$openfilter`([bOpen])

opens or closes the filter area if the grid has one, and returns `kTrue` if the operation was completed. `bOpen`: Use `kTrue` to open the filter area or `kFalse` to close it. Defaults to `kTrue` if unspecified.

JS List Control

Line Selection

The behavior for multi-select lists has changed when selecting and de-selecting list lines *and* when the Shift key is pressed. The change applies to all lists including the standard JS List control, Data Grid, Tree list, and Native List, when multiple line selection is enabled.

In addition, the `$keyboardchangesline` property now takes effect when `$multipleselect` is `kTrue` (Data Grid and Tree list only). In previous versions, when both properties were set to `kTrue`, `$keyboardchangesline` did not have any effect regardless of its state. This was so non-adjacent lines could be selected with the keyboard. This change allows multiple rows to be selected while also having the keyboard change the current line. To enable users to select non-adjacent lines with the keyboard, `$keyboardchangesline` can be set to `kFalse`.

JS Device Control

Multiple SMS recipients

You can now send a SMS to multiple recipients by assigning a comma-separated list of phone numbers to `$communicationaddress`. For example:

```
Do cinst.$objs.device.$communicationaddress.$assign(
    "0123456789,0987654321,0192837465")
```

Remember that the Device control only works when it is contained in a JavaScript wrapper.

Image Aspect Ratio

The JS Device control has a new property, `$imageaspect`, to allow the aspect ratio of a photo to be specified; it only affects images taken with the `kJSDeviceActionTakePhoto` device action (not `GetImage`). This functionality is only available in the iOS and Android wrappers, version 3.1.0 & later; also note the minimum Android version is now API21 (5.0, Lollipop).

The `$imageaspect` property takes a floating number, indicating *width* divided by *height*. If set to 0, no aspect ratio will be enforced, and the standard camera application will be used for taking photos. If greater than zero, a custom camera view within the app will be used, which shows the preview stream in the specified aspect ratio, and an image of the specified aspect will be returned. A value of 1 will enforce a square image.

The `$imageaspect` property can be used in conjunction with `$imagemegapixel` to take an image of specific dimensions, that is:

$$\text{\$imageaspect} = \text{targetWidth} / \text{targetHeight}$$

$$\text{\$imagemegapixel} = (\text{targetWidth} * \text{targetHeight}) / 1,000,000$$

The device control also has a new client-executed method, `$takephoto(iWidth, iHeight)` to provide a shorthand way of taking a photo with specific dimensions.

JS Droplist & Combo box

\$extraspacespace

The \$extraspacespace property has been added to both the JS Droplist and JS Combo box. The property is a number of pixels (≥ 0) that adds extra space to the lines in the dropped list box. (Note \$extraspacespace also applies to JS List control, Tree list, and Hyperlink controls.)

If \$extraspacespace is zero, the height of each row is the default height of the row content. If \$extraspacespace is greater than zero, the height of each row is the font height + \$extraspacespace.

\$borderstyle

The \$borderstyle property for Droplists, Combo boxes (and JS Edit controls) has been renamed to \$inputborderstyle, and is a kJSInputBorderStyle... constant that controls the appearance of kJSborderDefault, and which specifies the appearance of the control border when the control has the focus.

JS Complex Grid

Scrollable footer

The Complex Grid control can now include a scrollable footer section similar to the existing scrolling header section (this is also available in the window class complex grid control).

To enable a scrollable horizontal footer, you need to set \$showhorzfooter to true. The complex grid has the following properties to control the appearance of the footer:

- \$horzfooterheight**
The height of the grid horizontal footer
- \$horzfooterfillcolor**
The fill color for the grid horizontal footer
- \$horzfooterborder**
The border style for the grid horizontal footer
- \$horzfooterlinestyle**
The line style for the grid horizontal footer

Resize Row Animation

When you resize a row in a Complex Grid, you can now specify an animation type and duration for the resizing action. The method \$setrowheight(...) now takes two additional optional parameters: an ease constant, such as kAnimationCurveEaseOut, and a duration in milliseconds for the animation.

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.

JS Button

Border Appearance

The JS Button control has a new property, \$borderwidth, that specifies the border width in pixels (the default is 0 or no border). You can set the border color using \$bordercolor.

Flat button style

The style for all new buttons (and buttons in converted libraries) is now flat, so `$isflat` is set to true. In addition, if the value of `$buttonborderradius` in converted libraries is set to 0, it will now be changed to the new default of 4; any other value will be retained on conversion.

Disabled appearance

The appearance for disabled buttons has been improved, that is, when `$active` becomes `kfalse`. If `$isflat` is `kTrue` (the default), the button back color will become transparent (if it isn't already) and the text color will take on the `disabledText` color. If `$isflat` is `kFalse`, the button back color will take on the `disabled` color and the button text color will take on the `disabledText` color.

In addition, if `$bordercolor` for buttons is set to `kColorDefault` the color will match `$textcolor`. When disabled (`$active = kFalse`), the border will match the disabled text color to maintain a consistent disabled appearance.

JS Bar & Pie Charts

Theme Colors

You can now specify theme colors and explicit RGB integer colors for JS Bar chart & Pie chart segments.

The `$colorlist` runtime-only property can now contain `kJSThemeColor...` constants to allow you to match the colors in the current theme. In addition, you can specify an RGB integer using the `rgb()` function.

Note you cannot use standard Omnis color constants (such as `kRed`, etc.) in this context, since these are taken as literal text on the client.

Text and Axis Colors

The `$textcolor` and `$axiscolor` properties have been added to Bar Charts (Pie charts had `$textcolor` in previous versions). Both controls now use theme colors for this `$textcolor` which applies to the color of the title, labels, axis text, and legend in the chart, where applicable.

The `$axiscolor` property for a Bar chart applies to the color of the both axes lines, and the unit lines which run across the bar chart.

When set to `kColorDefault`, both properties will set their color dynamically according to the color of `$backcolor`.

In addition for Bar charts, when `$showvalue=kTrue`, the popup label will use `$backcolor` for the text and `$textcolor` for the background of the label so that it can be seen against the background of the control.

JS Tab Control, JS Segmented & JS Page Control

Current Tab, Segment & Page color

The properties `$currenttabindicatorcolor`, `$focusedsegmentindicatorcolor`, and `$currentpageindicatorcolor` have been added to the JS Tab controls, JS Segmented, and JS Page Control respectively, which are colors to indicate the current tab, segment, or page.

JS Popup Menu

Line Height

A new property `$menulineheight` has been added to remote forms to control the line height for all the menus in a remote form, including context menus and menus belonging to controls such as Popup, Tab strip and Splitbutton.

For existing applications, the value of `$menulineheight` will be zero, meaning the font size will determine the line height, as previous versions. For new applications, this will be a touch-friendly value to give enough space for each menu option.

JS Check Box, Radio Group & Switch

Color properties have been added to the JS Check box, Radio group and Switch controls so you can set their colors; note these can be theme colours so will change with a change of theme.

- `$checkboxcolor`
Color for the Check box control, and check boxes when they appear in Lists, Data grids & Tree lists.
- `$radiobuttoncolor`
Color for the Radio group control.
- `$switchcolor`
The `$switchcolor` property specifies the color for the Switch control when it is switched on (set to value 1), assuming no on/off icons have been set.

Event Method Validation

Omnis now validates the event codes you have entered when adding or editing *On* event commands in the Code Editor. Therefore, Omnis will check to see if the event code is valid for the current object, and if not, it will flag it as an error.

For remote forms, if the event is not specified in the `$events` property, Omnis will add it to `$events` automatically when editing a method named `$event` in a non-inherited object (Omnis displays a temporary status bar message when it does this).

You can turn off this validation using the **validateEventsForOnCommand** entry in the `methodEditor` group of `config.json`; set it to `false` to turn off event method validation.

Tab Order

All JS components now have a `$taborder` property in design mode (which is read-only) which shows the resolved tab order within the form, taking into account container fields, such as paged panes.

The context menu on a remote form includes the "Show `$taborder`" option (previously it was "Show `$order`"), so that you can see the value of `$taborder` for all controls on the form.

You can still alter the tab order of the controls in a form by modifying `$order` for each control.

The `$inheritedorder` property has been removed from the Property Manager for remote forms (it is still shown in the Notation Inspector for remote forms).

This property is set to zero by default and you are recommended to keep it set to zero which means that the designed order from the base class will be maintained.

Next Tab Object

All JS Components now have a property `$nexttabobject` which allows you to override the default tab order set by the `$order` property for all the controls in a remote form. The `$nexttabobject` property allows you to specify the name (`$name`) of the control you want the end user to tab to after the current object, overriding the tab order set by `$order`. You should not overuse this property, as it does incur some overhead by setting up additional event listeners.

Paged Pane

The behavior of the `evUserChangedPage` event has been modified. When the Paged Pane is linked to a Tab bar control, the `evUserChangedPage` event is triggered in the paged pane control when a tab is clicked to change the pane.

Control-level Return Methods

Support for control-level Return methods has been added. In previous versions, you were able to create a `_return` method in the remote form to return a value from the server to the client (you create a `'$serverMethod_return'` client-executed method in the form to receive the value returned from the server). Now you can create such return methods for controls (fields).

If a client-executed method calls a server-executed method on a control, when execution returns to the client, Omnis will look for the `'$serverMethod_return'` method on the control, and if not found it will look for the return method at the form level, as in previous versions.

JS Control Variable Names

The Property Manager now displays an error message when you try to assign an invalid data name property for a JS client object. This applies to `$dataname` as well as other similar properties such as `$listname` which require a variable.

Property Values in Client Methods

You can only read the value of a property in a JS client form/control instance when running in a client-executed method. The error reporting has been improved if you try to do this from a server-executed method. The improved error message when trying to read a remote form instance property on the server is now: "Cannot get the value of a remote form instance property when executing code on the server".

JavaScript Forms

The following new features and enhancements are for JavaScript Remote forms.

Subforms

Subform Dialogs

Two new *client commands* have been added to remote forms to allow you to open a single subform as a modal dialog: **subformdialogshow** opens the modal subform dialog, and **subformdialogclose** closes the topmost subform dialog.

The “**subformdialogshow**” command opens a single subform as a modal dialog.

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

Where `row` is `row(classname, params, title, width, height, closeButton, resizable, maxButton, openMax)`. The parameters are as follows:

- classname**
String, the name of the remoteform
- params**
String, literals to pass to the subform
- title**
String, the title of the modal dialog
- width**
Integer
the width of the dialog
- height**
Integer, the height of the dialog
- [closeButton]**
Boolean, defaults to true, show close button
- [resizable]**
Boolean, defaults to false, if true allows resizing

- ❑ **[maxButton]**
Boolean, defaults to false, if true shows maximize button (resizable must be set to true)
- ❑ **[openMax]**
Boolean, defaults to false, if true opens dialog in a maximized state (resizable must be set to true)

This command generates a new subform set and adds one modal subform to it. The name of this set is internal only, and cannot be added to or removed from. Another modal subform dialog can be opened above the previous one by running subsequent calls, preventing access to the first one until the second is closed.

The "**subformdialogclose**" command closes the topmost subform dialog. This command only works for subforms opened using the subformdialogshow command, and has no parameters as such modal dialogs must be closed in reverse order of them opening.

Subform Dimensions List

When you open a set of subforms in a *subform set* using the subformset_formadd client command, you can now pass dimensions for the subforms *for each breakpoint* in a responsive remote form; in previous versions you could only provide one set of dimensions.

A list can now be passed instead of the single set of left, top, width, height parameters in the subform client commands providing values for as many breakpoints as required. This works in both the formlist contained within the "subformset_add" and the individual "subformset_formadd" client commands. The same method applies to both client commands, but the example below shows directly adding a form with "subformset_formadd":

```
Do lDimList.$define(lBreakpoint,lLeft,lTop,lWidth,lHeight)
Do lDimList.$add(310,10,10,200,200)
Do lDimList.$add(600,kSFSCenter,kSFSCenter,300,300)
Do lDimList.$add(1000,kSFSCenter,kSFSCenter,600,600)
Do $cinst.$clientcommand(
    "subformset_formadd",row(
        cSetName,vUniqueID,cParams,cTitle,lDimList,iModal))
```

Note that the new dimensions list has replaced 4 separate parameters, and so has condensed the command to a minimum of 5 parameters (6 if passing a value for iModal). You can pass the parameter list accepted in previous versions or the new style list. However, this only works for responsive forms, whereas single and screen type forms must use the original set of parameters to avoid confusion.

The client will use the value passed in lBreakpoint to assign the values to the correct breakpoint for the containing form. If the breakpoints do not match, then the values will be used from the next breakpoint down. For example, if you had the list of dimensions as defined above, but your form used the following breakpoints:

310 - would use the values from 310 as they match

590 - this is smaller than the next value of 600, so would again use the values from 310

900 - this is smaller than the next value of 1000, so would use the values from 600

1200 - this is greater than the values from 1000, so uses those values.

\$loadfinished method

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, so you could create a client method called \$loadfinished to perform any actions you want after all subforms have loaded.

Control Menus

The `pControlMenu` event parameter has been added to the `evOpenContextMenu` and `evExecuteContextMenu` events to distinguish between events generated by a Control menu or a Context menu opened when clicking on the control.

All controls with a menu (Tab, Popup menu, and the new Split button) generate `evOpenContextMenu` and `evExecuteContextMenu` when using their own menus. To distinguish between Control menus and Context menus, a new boolean parameter, `pControlMenu`, has been added to both these events for all controls; `pControlMenu` is `kTrue` if the menu is a control menu, or `kFalse` if it is a context menu (JavaScript client only).

The `pControlMenu` event parameter has been added to the `evOpenContextMenu` and `evExecuteContextMenu` events generated by the Split Button control to allow you to detect events in the Control menu.

Form Layout Type

When designing a remote form, if you change `$layouttype` to `kLayoutTypeSingle`, and the `$resizemode` property is set to `kJSformResizeModeNone`, then `$resizemode` will be set to `kJSformResizeModeFull` automatically to make it resizable.

Remote Tasks: \$order

The `$order` property for remote tasks is not new but there was not an adequate description of the property in previous versions, so the property description and online help for `$order` of a remote task instance have both been updated, as follows.

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

PDF Printing

In previous versions, the Python files that are used to print a report to PDF were discarded. A new option `kDevOmnisPDFParamKeepScriptAndPNGs` has been added for the PDF device to allow you to keep the Python script and PNGs after printing to PDF. Use this constant with `$cdevice.$setparam` as the parameter number of the keep Python script and PNGs device parameter.

Runtime & Server Logging

There is a new Library preference, `$clib.$prefs.$alwayslog` (defaults to `kFalse`) to allow you to log messages in the Runtime and Server versions of Omnis to help you debug your code. When `kTrue`, the *Send to trace log* command and `tracelog()` function always write non-diagnostic messages to the trace log (overriding the check for debuggable code). In previous versions, the trace log recorded such messages in the Development version only.

Method Editor

The following new features and enhancements are for the Method Editor and Code Editor.

Code Folding

The Code Editor now supports *Code folding* which means you can *fold* and *unfold* (collapse and expand) blocks of code in order to assist with readability and code manipulation in general. If a code block can be folded, a '-' icon appears in the margin at the start of the block: when a block has been folded a '+' icon is shown next to the first line of the block, and directly under this is shown a "badge" (an ellipsis icon) representing the hidden code content.

The Code Editor shows a fold icon (☐) in the left margin which shows that a code block can be folded: you can click on the icon to fold the block, and the icon will toggle to show an unfold icon (⊕) to show that the block can be unfolded. For example, this is a code line before code folding:

```
☐ 1 If sys(6)='N'  
2   Calculate #S1 as 12345  
3   Send to trace log This is a test  
4 End If
```

When the mouse is over the fold icon, Omnis highlights the block that will be folded, for example:

```
☐ 1 If sys(6)='N'  
2   Calculate #S1 as 12345  
3   Send to trace log This is a test  
4 End If
```

After you have clicked the fold icon, and the code has been folded, the content is shown as a badge (ellipsis) representing the content of the folded block:

```
⊕ 1 If sys(6)='N'  
   ...  
4 End If
```

When the mouse is over the badge icon, Omnis displays a tooltip to show its content (this is like the method content tooltips already in Studio 10.1), but note that this tooltip is always displayed, irrespective of the Show Method Content Tips option. For example:

```
⊕ 1 If sys(6)='N'  
   ...  
4 End If 2 Calculate #S1 as 12345  
5          3 Send to trace log This is a test  
6
```

Just like method content tips, pressing the Shift key while the tooltip is displayed locks it in place until you remove the Shift key and move the mouse away. You can select the text in the tooltip and copy it to the clipboard.

You can also press the Control (Windows) or Command (macOS) key while the mouse is over a fold or unfold icon. In this case, if the command has multiple blocks that can

be folded or unfolded, Omnis highlights all the affected blocks, and pressing the fold or unfold icon while all blocks are highlighted opens or closes all the highlighted blocks. For example:

```

+ 1 If sys(6)='N'
    ...
+ 4 Else If
    ...
- 7 Else
  8 Calculate #S1 as 12345
  9 Send to trace log This is a test
10 End If

```

Code folding is only available in a block when there are *at least two method lines*: for a block that has a single line only, folding is not enabled for the block, so the folding icons are not shown, and the options in the folding menu are disabled.

Which Commands can be folded?

The following Omnis commands can be folded:

- All If commands, folded until the next Else, Else If or End If command in the same block.
- Else, folded until the next End If command in the same block.
- All Else If commands, folded until the next Else, Else If or End If command in the same block.
- All While commands, folded until the terminating End While command of the block.
- Both For commands, folded until the terminating End For command of the block.
- Repeat, folded until the terminating Until... command of the block.
- Switch, folded until the terminating End Switch command of the block.
- Case, folded until the next Case, Default or End Switch command in the same block.
- Default, folded until the next Case, Default or End Switch command in the same block.
- Begin reversible block, folded until the terminating End reversible block command of the block.
- Begin critical block, folded until the terminating End critical block command of the block.
- On and On default, folded until the next On or On default command, or the end of the method if there is no such command.

Code folding menu

In addition to using the fold or unfold icons in the left margin, you can use the fold/unfold options on a new Code folding menu, that can be used when the code editor has the focus. In this case, most of the menu items apply to the block containing the single line of code that is currently selected.

The Code folding menu is present on the Modify menu of the Method Editor and the Remote Debugger window for a remote debugger edit session. For a remote debugger debug session, there is a new Code menu on the toolbar, containing the Code folding menu commands.

The menu commands are:

Menu command	Description
Fold Block	Equivalent to pressing the Fold icon to fold the block.

Fold Block And Related Blocks	Equivalent to pressing the Fold icon while holding the Control (Windows) or Command (macOS) key to fold the block and other related blocks that can be folded.
Unfold Block	Equivalent to pressing the Unfold icon to unfold the block.
Unfold Block And Related Blocks	Equivalent to pressing the Unfold icon while holding the Control (Windows) or Command (macOS) key to unfold the block and other related blocks that can be unfolded.
Unfold All Blocks	Unfolds all folded blocks in the method.

The menu items also have shortcuts:

Windows

Fold Block	Alt+↑
Fold Block And Related Blocks	Ctrl+Alt+↑
Unfold Block	Alt+↓
Unfold Block And Related Blocks	Ctrl+Alt+↓
Unfold All Blocks	Alt+O

macOS

Fold Block	⌘↑
Fold Block And Related Blocks	⌘⌘↑
Unfold Block	⌘↓
Unfold Block And Related Blocks	⌘⌘↓
Unfold All Blocks	⌘⌘O

You can configure the keys for these shortcuts using the keys preference item, in the `methodEditorAndRemoteDebugger` group (in the `keys.json` file):

Preference item	Key(s)
<code>codeFold</code>	Opt + Up Arrow
<code>codeFoldRelated</code>	Cmd + Opt + Up Arrow
<code>codeUnfold</code>	Opt + Down Arrow
<code>codeUnfoldAll</code>	Cmd + Opt + O
<code>codeUnfoldRelated</code>	Cmd + Opt + Down Arrow

Selecting Code using the pointer

You can select the badge representing a code folded block, either using the keyboard or using the mouse. When the badge is selected, the content of the block it represents is selected. In addition, double clicking on the badge selects its content.

When Omnis needs to select a line in a folded block, e.g. when hitting a breakpoint, or clicking on a stack list entry, the editor *automatically unfolds the block* (and any containing blocks) in order to display the line correctly.

Entry Behavior

As soon as an edit would affect a folded block, Omnis automatically unfolds the block (and any containing blocks) before applying the edit.

Saving the Code Folding State

Omnis stores the code folding state with the method.

When using the method editor, the state is saved back to the class with the method, provided that the editor is not operating in read-only mode. In the latter case, you can still fold or unfold methods in a read-only class, but changes to the code folding state are not saved to the class.

When using the remote debugger, changes to the code folding state are saved locally to the cache of methods loaded from the server. However, once you re-open the debug session, these changes are lost; the one exception to this is any code folding that has been applied while editing a method in a remote debug edit session.

Therefore, you should consider code folding a semi-permanent state, since as soon as Omnis needs to display the contents of a folded block for some reason, it will open the block.

JSON Export

When Omnis exports a method as part of JSON export, it now appends the string \$... to the inline comment of commands that correspond to a code folded block. This allows Omnis to regenerate the code folding state of the method when it imports the class JSON.

Word Wrapping

Long lines of code displayed in the Code Editor will now wrap onto the next line automatically, and the text that wraps is drawn with an indent to make it clear that it belongs to the wrapped line (you can disable this behavior, so code lines are not wrapped, which corresponds to behavior in previous versions).

There is a new menu command, **Word Wrap**, on the View menu of the method editor and remote debugger windows to toggle Word wrapping; the option is turned on by default, and the state is saved with the window setup. When Word Wrap is enabled there is no horizontal scrollbar in the code editor window and long code lines wrap to the next line at suitable break characters, or they wrap if there is no break character.

For method content tooltips, word wrapping is always on, irrespective of the setting in the window for which the tooltip is being generated.

Inline comment wrapping & color

When word wrap is turned on and the Code editor encounters an *inline comment*, it tries to shrink the gap between the end of the code line and the inline comment *to avoid wrapping the code line if possible*: if the inline comment is still too long to fit onto the line it will wrap onto the next line, under the code line and is displayed indented.

As a result of these changes to the wrapping behavior of inline comments, their default color has been changed to gray for all of the themes (so they are different to code).

Method Search

There is a new **Search** or filter option in the method editor tree (method list) in the Method Editor (and Remote Debugger) to allow you to find specific named methods, or methods that start with or contain specific characters. As you type in the search box, the method list updates automatically to highlight the method names that match or contain the search (in currently expanded nodes only). These lines draw in a new colour, **treelinesmatchingsearchcolor** in the IDEMethodEditor section of appearance.json. The editor selects the first matching method for the search and shows its contents. While the search box has the focus, you can use the find and replace menu of the method editor (or its find next and find previous shortcuts) to select the next or previous matching method. There is also a new context menu item for the method list called "Select Found Methods", which selects all matching methods. There is a new menu option 'Search Method Tree' on the Find and Replace menu that puts the cursor in the method search box, which also has a keyboard shortcut named "searchMethodTree" that appears in keys.json - note that the default disable all breakpoints shortcut has changed as a result of this change.

The savePropertySearchDelay item in the ide section of the config.json file has been renamed saveSearchDelay, and now applies to both property and method name searches.

There is a new item on the View menu, "Show Method Tree Search Box", that allows you to toggle the method search box (the default is enabled). The state is saved with the window setup.

Showing Built-in Class Methods

It is now possible to view the built-in methods for a class in the code editor, making it easier to view their parameters and override them, if required. There is a new option in the View menu in the method editor, **Show Built-in Class Methods**, which allows you to toggle the option. When checked (the default), the class methods node in the method name list includes the *built-in class methods for instances* of the class type being edited, including \$control, \$construct, and \$destruct – this is the case for remote forms, remote tasks, window classes, and other classes that can be instantiated. In addition, \$scanclose will be shown for the relevant instance types, while \$select and \$fetch are shown for table classes. Many other methods could be shown depending on the class type, including \$filereadcomplete, \$init, \$term, \$sfsorder, \$sfscanclose, \$pushed, \$sqldone, \$suspended, \$resumed, \$loadfinished, \$previewurlclicked, \$pdfcomplete.

The built-in methods behave in a similar way to inherited methods, that is, you can override them, or set them back to using the default, by using "Built-in Method..." option from the menus (this is analogous to using Inherit Method... for an overridden inherited method). When you override a built-in method, Omnis pre-defines the parameters of the new method to match those required by the built-in method.

The names of the built-in class methods are shown in the tree using the no set property color (this is consistent with how built-in method names are drawn in the Interface Manager).

There is a new theme member **overriddenbuiltinmethodstyle** that can be used to give the name of an overridden built-in method a different text style when it is shown in the tree. This new theme member is in the IDEmethodEditor group of the appearance.json file, and can have the same possible values as **overriddenmethodstyle**; it defaults to 2 (italic).

Remote Debugger

You can now edit methods and code that you are debugging in the Remote Debugger. In Studio 10.0 & 10.1 you could debug and step through code on a remote server app, but you could not make any changes to the code: however, in this version you are able to make limited changes to your code.

The Remote Debugger client now has two options for opening a session: either **Open Debug Session**, or **Open Edit Session**. A "Debug session" works just like 10.1, so it can be used for remote debugging.

An "Edit session" allows you to edit methods via the remote debug interface window, that is, you can apply edits, and you can create new variables via the fix error dialog. Note that until you try to save the method back to the server, you will not know for sure if the method will be accepted, since only part of the library is available when editing - you can use instance, class, local, task and parameter variables (from the class or a superclass) or any file class variable in a file class used by the method. Using variables from other file classes, or using notation, functions or commands available on the server (but not the client), will be displayed as an error if you edit a line containing something only available on the server. However, you can still save the method successfully in this case.

In edit mode, methods default to read-only in the remote debug window. You need to explicitly press "edit method" in the toolbar to edit the method, after which you cannot do anything else with the window until you press Save or Cancel or close the window.

Remote Debug Menu

There is a new Omnis preference \$showremotedebugmenu (\$root.\$prefs) to control whether or not the Remote Debug menu is displayed. It defaults to kFalse, and is not saved. Therefore, if you want a library to display the Remote Debug menu, you must assign kTrue to this property in your startup code.

Server Port

The default server port for the remote debugger ("debugPort" in the config file) is now 6102.

Resolved Name Colors

New syntax colors and styles have been added to the Code Editor to highlight field names and parameters that are "resolved" or "unresolved" for certain commands that reference field names and notation group members.

The Code Editor can now optionally (defaults to on) display names it has resolved using a new **resolvednamecolor** and **resolvednamestyle**, and names it has failed to resolve using **unresolvednamecolor** and **unresolvednamestyle**.

If `useresolvednamecolorsandstyles` is true (all members are in the `IDEMethodSyntax` section of `appearance.json`) the code editor tries to resolve certain names, and if successful draws them using the resolvedname color and style; if unsuccessful it draws them using the unresolvedname color and style.

Examples of where this applies are the parameters of the *Redraw* command, *Queue set current field* command, names in notation such as `$cinst.$objs.name`, and method names in calls such as `$cinst.$mymethod()`.

If a name is drawn using the unresolvedname color and style it does not necessarily mean there is an issue, e.g. it could be a notation reference such as `$cinst.$objs.name`, where the object is dynamically added and named at runtime.

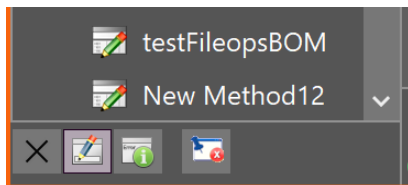
Appearance Colors

There are some new colors in the Code Editor, defined in the `IDEMethodEditor` section of `appearance.json` file (and included in the theme files):

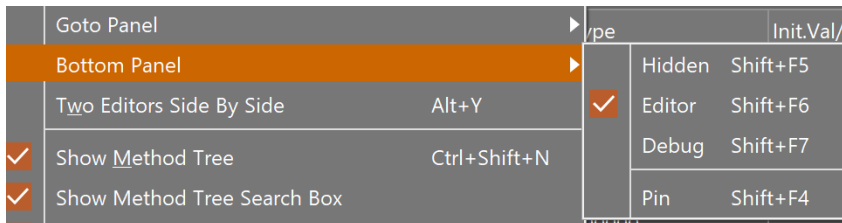
- codeassistantpopupcolor**
The background color of the Code Assistant popup window.
- treelinesmatchingsearchcolor**
The background color of unselected method editor tree lines that match the current method search.
- methodeditorcodeleftmarginbackgroundcolor**
The background color of the left margin of the code editor (where the Go point and breakpoints are shown).
- overriddenbuiltinmethodstyle**
The background color of an overridden built-in method in the method list
- executionpositioncolor**
lines are drawn *above and below* the Go point code line and Call stack return lines using the new color
- methodeditorcodereadonlybackgroundcolor**
background color showing that a class is read-only and therefore its methods cannot be edited
- styledbadgebackgroundcolor** and **styledbadgetextcolor**
The background color and text color of badges drawn in styled text in the Code Folding in the Code Editor, and defined in the `IDEGeneral` section of `appearance.json`. A badge is also shown on the Trace Log node of the Studio Browser tree, to show the number of trace log lines.

Panel Popup Menu

The Panel Popup menu, previously underneath the Code Editor area, has been moved to the lower left corner of the Method Editor window, below the method name list, but otherwise the buttons perform the same action.

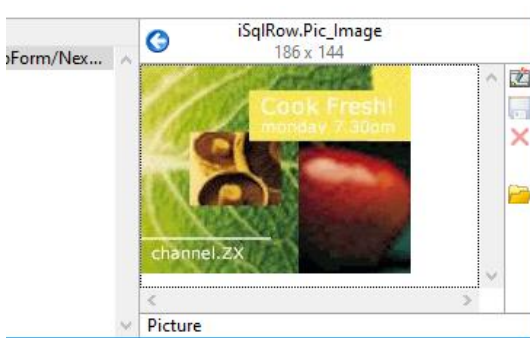


The equivalent options are on a new hierarchical menu called Bottom Panel on the **View** menu in the Code Editor.



Save Image in Debugger

You can now save an image from the debug variable panel in the Code editor using a new Save picture button (folder icon). The new button is available when viewing image data in the debug variable panel, in the modify tool strip to the right of the image.



The Save picture button is enabled when not modifying the variable value, and when the debugger recognizes a JPEG, GIF or PNG (the latter includes shared pictures stored as PNG, in which case the saved image is a PNG without the shared picture header).

The new button uses the binaryEditOperations keyboard shortcut.

To line

You can now Alt-click in the left margin of the Code Editor to execute the debug command "To line" provided that code is executing.

Do and Quit method commands

Normally, all commands matching the first typed character appear in the Code Assistant list, but you can limit or change which commands are shown, or selected by default if there is only one matching command, by enabling the **Use Minimum Lengths** option on the Filter Commands submenu (note the state of this option is saved in the Save Window Setup). This option is now enabled by default, meaning that the *Do* command will be selected by default when you type 'D' (rather than the *Default* command), and *Quit method* will be selected by default when you type 'Q' (rather than the *Queue* commands).

Variable tips

Precedence is now given to variables over functions when generating tooltips for the Code Editor, for example, when a variable name is the same as a function name (although this is generally not recommended).

Documentation tab

You can now change the width of the fields on the documentation tab in Code Editor by dragging their borders.

The positions are not saved, and will revert to equal distribution when resizing the code editor or changing method.

Boolean Variables

Omnis no longer treats empty and false as two different values of Boolean variables, when displaying them in the debugger. Therefore the debugger variable panel, variable tooltips, variable context menu and variable window now all display and treat empty as False or NO as appropriate.

Copy Lines

There is a new option in the Code Editor context menu, Copy Lines, to allow you to copy the *complete code* in the current line (the line containing the caret), or *all complete lines* in the current selection.

Select Object

A search box has been added to the Select Object dialog (opened when you select the subtype of an object or object reference variable). A Search box has also been added to Set Superclass dialog.

Sta: command and Square Brackets

In previous versions, there was a problem entering quoted square bracket expressions in the Sta: command, which has been fixed in this version, but as a result of the fix, the *close square bracket* (]) is now not added automatically when editing a Sta: command.

When you split a text block command parameter using Return (carriage return) the "Sta:" command prefix is now inserted into the text block automatically.

Text: and parenthesis

In previous versions, the Text: command with just an open bracket (as its parameter would not tokenize and caused an error. Therefore, the way the Code Editor handles (at the end of a code line has been modified.

If Omnis encounters (at the end of a command line, it prompts for options (Carriage return etc). If there is another character after the (, without a trailing comma, Omnis stops looking for options, and treats the characters as text. This leaves the special case of (on its own at the end of the text. You can enter this using square bracket notation with a constant [kOpenParen]. There is also a new kCloseParen constant.

Omnis Help

The behavior of the inline Omnis Help system after pressing F1 or using the Help menu while using the new Code Editor has been improved. The new behavior is as follows:

1. If no text is selected in the Code Editor, it tries to obtain the text from the syntax item containing the caret - if there is nothing useful, no help will be displayed, otherwise it will pass the text for the syntax item to the help system, e.g. 'Calculate' for a Calculate command when the caret is in the command name.
2. If some text is selected, and all selected text is on a single line, the editor passes the selected text to the Help system. If the selected text spans lines, no help will be displayed.

After performing 1 or 2, the Help system opens. If the text passed to the Help system uniquely identifies a single help page, that help page is displayed. Otherwise, the help window opens at the search tab, searching for the text passed to the Help system.

Help for Built-in Functions

In previous versions, the Code Assistant displayed a short text description for built-in functions, but now the full help page for the function is shown. There is a new entry `useOmnisHelpPagesForFunctionHelp` in the `codeAssistant` section of `config.json` which you can set to control this behavior.

If a help page does not exist for the function, or `useOmnisHelpPagesForFunctionHelp` is set to `false`, the code assistant reverts to showing the text description.

Auto tab: Table instance data

There is a new entry "Table instance data" at the start of the Auto tab when debugging code in a table instance. Simple references like `$cinst.name` will show in the Auto tab, when name is not a variable in the normal variable scopes, e.g. a column in a row in a table instance.

MultiProcess Server

Under normal operation, the multi-threaded Omnis Server does not take full advantage of multi-core processors, because it uses a *time-slicing model* that single threads the execution of Omnis code in all situations, other than when some sort of external call (e.g. a DAM call) is in progress. The concept of the **MultiProcess Server** (MPS) for the Linux Headless server has been designed to eliminate this short-coming and deliver significant performance improvements in your applications, by using a *multiprocess* rather than *multithreaded* server model.

When using the MPS in the Omnis Linux Headless Server:

- There is a *single main server process* that receives requests from clients.
- There is a *separate child process for each client*, represented by a single remote task.

The main server process passes the request to a child process which executes the request. The child process then passes its response back to the main process, which then sends the response to the client.

Each child process is created using a forking system; however, the server is implemented so that when a child process becomes free (because its remote task destructs), it can be added to a pool of free child processes, ready to be associated with a new remote task. This greatly improves performance.

One of the main features of the new MPS is that it can be plugged into an existing server configuration, and it will still work with the load sharing process; in other words, it still has exactly the same interface via its server port.

Another major advantage of using the MPS is that since execution is isolated to a single client per process, any problem in the child process (a crash perhaps), will only result in a single client receiving an error, and the server will continue running.

With the implementation of the MPS, there is some new functionality and some changes to existing functionality, with regards to opening libraries, and the way class data is cached, which are described below.

Configuration

To use the MultiProcess Server (MPS), you need to add (or enable) some new entries in the "server" section of the server configuration file (`config.json`) for the Linux Headless Server. The new entries are:

- multiProcess**
When `multiProcess` is true, the Linux Headless Server will start up in `multiprocess`

mode; in this case, the entries start, stacks and timeslice in the server section of config.json are ignored as they are not relevant.

- ❑ **maxChildProcesses**
is the maximum number of child processes.
- ❑ **maxFreeChildProcesses**
is the maximum number of free child processes (not associated with a remote task).

The new options are written to the server section of config.json like this:

```
"server": {
  "multiProcess": true
  "maxChildProcesses": c,
  "maxFreeChildProcesses": f
},
```

In addition, the headless server in all its variants (single-threaded, multi-threaded and multi-process) supports some new entries in the server section of config.json that provide some control over reading requests from a client:

```
"timeoutReads": true,
"readTimeout": s
```

These entries indicate if the server will timeout a connection from a client if the complete request is not received in readTimeout seconds.

Configuration files

Child processes never write to the files omnis.cfg and config.json.

Libraries

The MPS starts up just like the normal headless server. As such, it opens libraries in the startup folder, and constructs their startup tasks. There are however some rules that need to be followed, because of the way the forking process works:

- ❑ *No DAM connections can be left open* after the startup task constructors have run.
- ❑ *No files opened by FileOps or other externals should be left open* after the startup task constructors have run. This is because their file descriptors will be shared by each child process, because of how the forking process works.
- ❑ *A child process can only write to a library that it has opened or created itself* (this is opened with exclusive access by the child process). If the child process attempts to save a class to a library it did not open or create, Omnis ignores the error and returns success rather than an error code.
- ❑ *A child process can only close a library that it has opened or created itself.*
- ❑ *osadmin cannot open and close libraries in the MPS.*

Internally, when the forking occurs, the child process closes and re-opens the file descriptor (read-only, shared) for all open library files, since the otherwise shared file descriptor with the main process has a common shared file offset. Additionally, byte range locking calls in the child become no-ops.

Classes

As part of startup of the MPS, Omnis caches all class data from all open startup libraries in memory. This allows the class data to be immediately available to a child process after it is created using the forking process. As stated earlier, you cannot write to the startup libraries. Therefore, you *should not modify classes belonging to these*

libraries in a child process, since the child process will typically be used for many remote task instances during its lifetime. However, this is not enforced.

Commands

You cannot use the following commands in the MPS:

- ❑ *Start server* and *Stop server*.
- ❑ *Set timer method* and *Clear timer method*.

All of the above generate the error 125446 (cannot use this command, function, or notation, in the multi-process server).

You cannot use the command *Quit Omnis* in a child process of the MPS. Attempting this generates the error 125437 (cannot use this command, function, or notation, when running in a child process in the multi-process server).

Finally, the commands *Begin critical block* and *End critical block* have no effect in the MPS. This is because each child process handles a single remote task.

New sys() functions

There are two new sys() functions:

- ❑ **sys(243)** returns true if and only if Omnis is running in MPS mode.
- ❑ **sys(242)** returns the child process ID, a character string that uniquely identifies the child process that is currently running. When the method is not running in the MPS, or not running in a child process, this has the value "0".

sys(242) can be used to identify the child process that is to process a request from a client: see the section "Using The Same Child Process" later in this document.

Process init method (\$processinit)

When the MPS creates a new child process via the forking process, the child process runs the \$processinit() custom method (if present) in the startup task of each open startup library. You can use \$processinit() to carry out any initialization required to set up the environment in which all remote tasks handled by the child process will run.

Database Connections

Each child process *has its own SQL database connections*. You could use \$processinit(), for example, to create a server pool containing a single database connection, that you can then use for all remote tasks that the child process handles.

Remote Task Methods

\$maxusers

The MPS does not support the \$maxusers property of a remote task.

\$sendall()

Due to its multi-process architecture, the MPS does not support notation such as \$iremotetasks.\$sendall(), because if you execute this in a child process, it will only apply to the currently executing remote task.

To overcome this, Omnis now includes (for all platforms, and all variants of server: single-threaded, multi-threaded and MPS), some new notation that allows you to "*broadcast*" a message to all remote tasks, including those running in another child process in the MPS.

Sending messages to Remote task instances using \$broadcast()

There is a new method of the \$iremotetasks group of remote task instances called \$broadcast() that can be used to send or "broadcast" a message via a public method to all task instances; its syntax is:

❑ \$broadcast()

`$remotetasks.$broadcast(cMethod, vListOrRow [, bWait=kTrue])`

Calls the public method `cMethod` in all open remote task instances, passing `vListOrRow` as a parameter to each call. If `bWait` is `kTrue` returns a list of return values, containing a line for each remote task that executed the method.

`cMethod` need not exist in all remote tasks; if it does not exist, Omnis ignores the remote task.

When using `bWait` with value `kTrue`, all remote tasks must return the same data type. If the returned type is row, then the return value list is defined to have all of the columns of the row (so all remote tasks must use the same definition for their returned row), otherwise the return value list has a single column with the returned data type as its type.

Omnis Data Files

Omnis data files cannot be opened in the MPS. Attempting to open one causes the error 101437 (Cannot use data files (either because the serial number does not allow data files or because Omnis is running as a multi-process server or because Omnis was invoked with `--runscript`)).

Icon data files such as `omnispic.df1` can still be used. As for libraries, the child process closes and re-opens (read-only, shared) the file descriptor for all open picture data files when it initializes itself after it has been created by the forking process.

Execution

When a new message arrives from a client, the main process inspects it. If the message is for an existing remote task, the main process sends it to the child process handling that remote task; to do this, the main process maintains a table that maps remote task connection id to child process. If the message requires a new remote task, then the main process either sends it to a free child process, or creates a new child process via the forking process (if configuration allows), and sends the request to the new child; if the configuration does not allow (the `maxChildProcesses` limit has been reached), the main process queues the request internally until a child process becomes available. This latter queueing behavior works best in a server handling RESTful, ultra-thin or SOAP web service requests, since these requests are usually processed relatively quickly; when the server is handling JavaScript client remote forms, it is best to allow essentially unlimited child processes, so a client can connect immediately.

Child processes send an event to the main process when their remote task destructs. The main process can then decide whether to tell the child process to exit (because the maximum number of free child processes has been reached), or add the child process to the pool of free child processes.

Licensing

Management of the server user count is handled using a shared memory semaphore. The main process initializes the semaphore with a count equal to the server user count. When a child process creates a remote task, and therefore needs a license, it waits on the semaphore. For a RESTful request, it waits indefinitely, and for other requests it tries to wait, and if the semaphore does not have availability, it rejects the request. This behavior is then equivalent to that of the multi-threaded and single-threaded servers, in that RESTful requests are queued waiting for a license, and other requests are rejected immediately if a license is not available.

When a child process deletes a remote task, it frees the license by posting the semaphore.

If a child process crashes while it is holding a license, the license will not be freed by the child. However, the main Omnis process attempts to restore the license semaphore

count by posting to the semaphore appropriately, based on the server user count and the number of child processes currently assigned to a remote task.

Load Sharing

It is possible to use the MPS in conjunction with the Load Sharing Process, although this only really makes sense when each Omnis server accessed via the load sharing process is running on a separate machine, since the MPS is essentially providing a load sharing mechanism. The main process maintains the load sharing statistics and responds to the load sharing statistics request message.

Remote Debugging

Due to the dynamic nature of the Omnis environment, remote debugging is supported in the MPS only in the context of a single child process, explicitly created to execute clients that are to be remotely debugged – this is called a *Remote Debug Child*.

Debugging Startup

The following sections describe key points regarding how to remotely debug Omnis code running in the MPS.

If the member **pauseAtStartupUntilDebuggerClientStartsExecution** in the remote debug configuration is true, you can debug the startup tasks of the remotely debuggable code in the MPS. This code runs in the main process.

Start the MPS, use a Windows or macOS copy of Omnis to connect a remote debug client to the MPS, and debug the startup code.

After MPS startup completes, the remote debug connection closes.

Debugging the Remote Debug Child

After MPS startup, assuming remote debugging is enabled, the MPS creates the remote debug child, and the remote debug child then becomes the instance of Omnis visible to remote debug clients. Therefore, using the remote debug client on a macOS or Windows copy of Omnis, you can connect to the MPS remote debug child, and debug that directly.

If the member **pauseAtStartupUntilDebuggerClientStartsExecution** in the remote debug configuration is true, the remote debug child pauses, waiting for a remote debug client to connect, before running any `$processinit()` methods. In this case, the remote debug client has a hyperlink “Run Process Init” rather than “Run Startup”.

Making a Client Use the Remote Debug Child

For JavaScript remote form clients, specify `?omnisRemoteDebug=1` on the URL used to open the Omnis remote form.

For ultra-thin clients, include `omnisRemoteDebug=1` in the query string used to invoke the ultra-thin request.

For RESTful clients, include the header `omnisremotedebug` in the request.

In all of the above cases, the request that requires the remote debug child will be queued if necessary, waiting for that child to become available.

Using the Same Child Process

The MPS allows you to cache data for ultra-thin and RESTful requests, to improve performance, using a query string parameter or HTTP header respectively, to specify the process ID (`sys(242)`) of the child process that is to ideally process the request.

When a request arrives that identifies a specific child process, the main process sends it to that child if the child still exists, otherwise it sends it to any available child.

For ultra-thin, the query string parameter is named `ProcessID`.

For RESTful, the HTTP header is named `omnisprocessid`. When using RESTful in conjunction with a process ID, you must always immediately respond to all requests, i.e. you cannot defer the response until later by assigning `kFalse` to `$restfulapiwillclose`.

Logging

The old-style web service logging to a data file is not supported for the MPS. Instead, you can configure the **datatolog** for the **logToFile** logcomp to include web services.

If you configure logging to go to standard output, by setting **stdout** to true in the `logToFile` object in `config.json`, logging from all processes in the MPS (main and child) will go directly to standard output, serialised between all of the processes using a shared mutex.

If you configure logging to go to a file in the logs folder, all child processes send their log data to the main process, which then writes the log data to file.

Command Line

All versions of Omnis on all platforms now use `--version` as the switch to report the Omnis version and build number, rather than `-version`.

The Linux Headless Server has a number of new command line switches. Pass the switch `--help` to display them all.

Command	Action
<code>homonis --help</code>	Print the help information and then exit
<code>homonis --version</code>	Print version and build number and then exit
<code>homonis <sw></code>	Start the server using the start server and multi-process settings in <code>config.json</code>
<code>homonis <sw> --st</code>	Start the single threaded server ignoring the start server and multi-process settings in <code>config.json</code>
<code>homonis <sw>--mt</code>	Start the multi-threaded server ignoring the start server and multi-process settings in <code>config.json</code>
<code>homonis <sw>--mp</code>	Start the multi-process server ignoring the start server and multi-process settings in <code>config.json</code>
<code>homonis <sw> --runscript=path <args></code>	Open the supplied library identified by <code><path></code> , construct its startup task, and pass the remaining command line arguments <code><args></code> to the <code>\$runscript</code> method in the startup task

`<sw>` can be any combination of the following:

Switch	Meaning
<code>--jscomments</code>	Includes comments in client-executed JavaScript generated by <code>homonis</code>

Switch	Meaning
--debugport=n	Overrides the configured remote debugging port
--pausestartup	Forces homnis to wait for a remote debugging client to connect before running startup (and \$processinit if relevant)
--debugscript	Starts the remote debug server at startup when invoked with --runscript
Any other user parameters that can be accessed from Omnis code using sys(202)	

--runscript

This mode allows you to use headless Omnis to run a script, that is, use headless Omnis to run some Omnis code within a shell script. For example, you could use the HTTP Worker Object in some Omnis code to invoke requests against an Omnis server. Used in conjunction with bash (or other shell) this can be quite powerful.

For example, you can write a script such as:

```
for i in {1..10}
do
    homnis --runscript=mylib.lbs <args> &
done
wait
```

and execute it using

```
time ./myscript
```

This will create 10 instances of the run script homnis process, wait for them to complete, and report how long execution took.

In more detail, the path passed via the runscript argument is the path of an Omnis library, the startup task of which must contain a method named \$runscript. This method receives as parameters the remaining parameters on the homnis command line, so for example you could pass a URL or an iteration count, or both. When homnis starts up in run script mode, it opens just this single library (ignoring startup libraries) and executes the method \$runscript in the context of its startup task. The script library is responsible for calling *Quit Omnis* when it has finished. This allows it to start workers which may not complete until after \$runscript has returned.

When homnis is running a script:

- It does not write to configuration files omnis.cfg and config.json.
- It does not accept command input from stdin.
- It only logs to stdout if logging is configured
- It will only open the script library passed as a parameter
- It will not open Omnis data files

You can use the new function printf() to output a string from a script: printf(string[, newline=kTrue]) writes the string to standard output followed by a newline character if required (Ignored on Windows. Executes on macOS and Linux only).

External Components

It is possible that a loaded external component will not be in a good state after the forking process, probably due to problems with data structures in use by external libraries it is using (typically data structures containing some sort of operating system handle or file descriptor).

To cater for this, an external component with this kind of issue needs to return the flag `EXT_FLAG_RELOAD_AFTER_FORK` in the flags returned by `ECM_CONNECT`. This means that the main process unloads the component (calling `ECM_DISCONNECT`) after startup completes. Each child process then reloads the component (calling `ECM_CONNECT` again) as part of its initialization. As a result, each child process has a clean copy of the component.

Window Components

The following new features and enhancements are for Window Class components (not JavaScript components or remote forms).

Token Entry Field

A *Token Entry Field* is a new type of text field for window classes (thick client) into which the end user can enter text which then becomes tokenized; a *token* is a single block of text that can be easily selected, moved, copied or deleted as a *single item*. The behavior of the Token Entry Field is very similar to the Recipients (To:) field in an email program, such as Apple mail or Google gmail, where you enter each email address as a block or token. When you start to type a name, a popup list will appear containing all entries that match what you typed and you can select one of the emails in the popup list, or you can complete the email address manually. You can then press Tab or Return to complete your selection and the text becomes a single block or token.

A token in the token entry field is defined to be a *character string* that either conforms to the syntax specified by a regular expression associated with the field (`$tokenregexp`), or matches one of a set of possible values in a list associated with the field (`$tokenlist`). As the token entry field operates on text, its `$dataname` needs to be set to a Character variable.

The token entry field allows the end user to enter and display a delimited list of tokens, separated from each other by a delimiter character (such as a comma, the default delimiter). Tokens also have an additional syntax (defined by RFC822), where they can be expressed in the form `displayText<tokenValue>`. In this case *tokenValue* is the actual token that conforms to the regular expression or a value from the token list, and *displayText* is the text displayed in the field when the item is tokenized (and not being edited). For example, consider the comma-delimited text string containing four tokens (email addresses):

Bob Mitchell<bob.mitchell@omnis.net>,Jason Gissing<jason.gissing@omnis.net>,Bob Whiting<bob.whiting@omnis.net>,colin.richardson@omnis.net

When displayed in the token entry field, it will look like the following, when the third token is currently being edited:

Bob Mitchell ▼ × Jason Gissing ▼ ×
 Bob Whiting<bob.whiting@omnis.net>
 colin.richardson@omnis.net ▼ ×

Using a Token Entry Field

As you start typing text into a Token Entry Field, the text will become a token value: if the field uses the additional display text syntax, then this needs to be part of the text you are entering. As soon as you press Return or a delimiter character, this terminates entry of the new token, the token is then displayed as either a valid token or an invalid token, and the caret is positioned after the token, ready for you to enter the next token.

valid token ∨ × **invalid token** ×

There is a down arrow button for valid tokens, which you can press to open a menu for the token, while there is an x button for invalid tokens that you can press to delete the token.

The token entry field does not allow empty tokens by removing consecutive delimiter characters. In addition, the token entry field does not support overtype mode, and does not allow leading or trailing whitespace for tokens and will strip these automatically.

Once you have entered some tokens, you can use the arrow keys to navigate around the tokens, and then insert a new token by typing some text if required. While you are entering a new token value, mouse selection of text, and left arrow, right arrow and other relevant key combinations are restricted to the new token being entered, to prevent premature termination of edit mode for the new token. However, up or down arrow will terminate entry of the new token.

When the caret is positioned between tokens, pressing forward delete will delete the next token, and pressing backspace will select the previous token (this latter behaviour is how the macOS standard token entry field behaves). In addition, when the caret is positioned before a token, pressing return starts editing the token.

The token entry field also supports popup assistance, providing a list of tokens that match the text entered so far, allowing you to select one of the values in the popup list to populate the new token. The popup only displays after a pause in typing, the length of which can be configured in config.json ("defaults", "tokenEntryPopupDelay", default value 500 milliseconds). If there is a single valid token selected, or if the caret is positioned before a valid token, pressing Alt/Option+Return opens the menu for the token if present.

You can select a token or contiguous range of tokens using the arrow and other keys, like any other objects, so that you can delete them or copy them to the clipboard for example. Double-clicking on a token (when the Shift key is not pressed) starts editing the token. The Undo option works as you would expect for an Entry field.

You can use drag and drop to re-order tokens, or to move or copy tokens from one token entry field to another. In order to do this, a couple of lines of Omnis code are required (described below).

Properties

The token entry field supports many of the standard properties supported by other entry fields. The \$objtype property for the token entry field has the value kTokenEntry. The \$dropmode property has a new constant kAcceptTokenEntry to accept drop data.

\$tokenlist

The \$tokenlist property is the name of the list variable containing all possible token values which is used for popup assistance and token validation. If omitted or empty, the list for popup assistance is obtained by sending the evGetTokenList event.

When specified, the \$tokenlist can have just one Character column containing all the valid token values, possibly including some displayText. The \$tokenlist can contain more than one column, in this case it must have a column with the name 'name' that contains all of the valid token values, possibly including some displayText. The list can optionally have additional character columns named 'desc' and 'iconid', that contain a short description of the token, and an icon to represent the token.

You may need to consider performance when deciding whether or not to use `$tokenlist`. Performance is affected by a combination of the number of possible token values, and the likely number of tokens entered.

The entry field popup will always display either the token text or `displayText`. In addition, the popup will display the `desc` value and the 32x32 icon with the specified icon id if either or both of these columns are present. The popup draws the `desc` column using the same font as the token entry field, with a point size 2 smaller (unless the token entry field font point size is less than 7, in which case it uses the same font). For example:



\$tokenregexp

The `$tokenregexp` property is a regular expression used to validate the syntax of a token. If `$tokenlist` has a list of all possible tokens, you can still use `$tokenregexp` for a pre-validation step that reduces searches of `$tokenlist`.

The regular expression in `$tokenregexp` uses the new PCRE2 implementation. If you omit both `$tokenlist` and `$tokenregexp`, all token values are valid.

\$tokencase

If true, tokens are case sensitive. This affects both searches of the token list and execution of the regular expression.

\$tokenmenu

The `$tokenmenu` property contains the contents of the token menu. If specified, all valid tokens have a dropdown button that can be used to open this menu (as a context menu). The selected token is available as the `pToken` event parameter of `evOpenContextMenu`.

You can distinguish between the user using the context menu and the user using the token menu, because `pToken` has the value `#NULL` when using the context menu.

Note also that `pToken` includes the `displayText` as well as the token value if the token has a `displayText` component.

\$showtokendeletebutton

If `$showtokendeletebutton` is true, all tokens have a delete button when the control is enabled.

3.7 \$tokendelimiters

The `$tokendelimiters` property contains one or more delimiter characters which are used to separate tokens (default is a comma). Each character specified can be used to separate tokens from each other in the data stored in `$dataname`. The first delimiter character is the default that the control uses when it needs a delimiter.

Note that this means that token values and `displayText` cannot include any of the delimiter characters, nor can they include `<` or `>`.

Token Colors

The token entry field has 4 token-specific color properties: `$validtokenbackcolor`, `$validtokentextcolor`, `$invalidtokenbackcolor`, `$invalidtokentextcolor`.

Each of these can be set to `kColorDefault`, which means use the corresponding entry in the token section of `appearance.json` (this is a new section for this control).

Events

The token entry field supports the same standard events as the multi-line entry field, with a few exceptions noted here. It does not generate `evClick`, and will only generate `evDoubleClick` when the control is read-only. In the latter case, if `$allowcopy` is `kTrue`, it will only generate `evDoubleClick` when double-clicking on a token.

As the control does not scroll horizontally, it does not generate `evHScrolled`.

evGetTokenList

The `evGetTokenList` event is sent to the token entry field when `$tokenlist` is not specified, to get the list of possible token values for the popup. This event has two event parameters:

- ❑ `pNewText`: The text entered by the user, for which the list of possible token values is required.
- ❑ `pTokenList`: The list of possible token values to be displayed by the token entry popup. Assign this parameter when processing `evGetTokenList`.

The list to return via `pTokenList` has the same characteristics as `$tokenlist`. Note that whereas the token field sorts the list of values to display in the popup when using `$tokenlist`, the token entry field does not sort the list returned via `pTokenList`.

A simple example:

```
On evGetTokenList
  Do pTokenList.$define(cTokens)
    For #1 from 1 to 10
      Do pTokenList.$add(con(pNewText, "Test", #1))
    End For
```

Methods

\$gettokens

`$gettokens([bSelOnly=kFalse, bIncludeInvalid=kFalse, bIncludeDisplayString=kFalse, bSort=kFalse, bRemoveDuplicates=kFalse])`

Returns a single column list of the tokens stored in the control.

- ❑ `bSelOnly`. Only return selected tokens.
- ❑ `bIncludeInvalid`. If true, return all tokens, otherwise just return valid tokens.
- ❑ `bIncludeDisplayString`. If true, the values added to the list include the `displayText` component.
- ❑ `bSort`. If true, sort the returned list in ascending order. Sorting uses `$tokencase` to determine if case-sensitive sorting is required.
- ❑ `bRemoveDuplicates`. If true, remove duplicate entries. Note that if `bRemoveDuplicates` is `kTrue` `bSort` must also be `kTrue`.

\$droptokens

`$droptokens([bRemove=kFalse, bSetFocus=kTrue])`

Call this during `evDrop` for the token entry field, to insert dragged tokens into the current drop location, optionally remove them from the drag source, and optionally set focus to the drop destination. Returns true for success.

- ❑ `bRemove`. If true, remove the dragged tokens from the drag source, if it is a token entry field.
- ❑ `bSetFocus`. If true, set the focus to the destination token entry field where the drop is occurring, after moving or copying the tokens.

You can set up the token entry field to accept data dragged from other controls, and if that data is text it can be used with `$droptokens()`; in this case, `bRemove` is not applicable. For example, in `$event` for a token entry field:

```
On evDrop
  Do $cobj.$droptokens(kTrue)
```

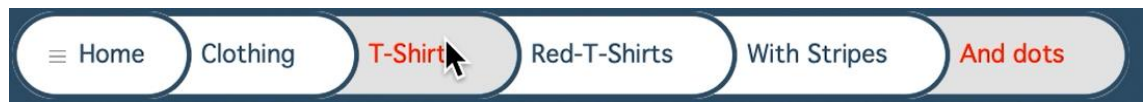

In this case, you need to set `$dropmode` to either accept all, or include `kAcceptTokenEntry`.

Breadcrumb Control

The Breadcrumb control is a new window class component (not available for JS client) which can be used to display the end user's "location" within the hierarchy of an application, as well as allowing the end user to navigate back up the system by clicking on one of the "crumbs" or the "Home" crumb. A breadcrumb control is often seen in the context of a website, as secondary navigation, but it can be used to enhance the UI for many types of desktop applications, such as consoles and dashboards.



The Breadcrumb control can be displayed in different styles: Arrow (as above), Rounded rectangle, or plain Text; this is set in the `$crumbstyle` property. The following is the rounded style, showing the third crumb highlighted when the pointer is over it.



The following shows the plain text style, showing the fourth crumb highlighted when the pointer is over it.



Properties

The content for the other crumbs in the control are taken from a list assigned to its `$dataname` property. The list has three columns: `crumb_text` (Character), `crumb_id` (integer), `crumb_icon_id` (Character) – the icon ID is the name of an icon in an icon set assigned to the library. The first crumb in the control is referred to as the "Home" crumb and is always visible; its text and icon are defined in the `$homecrumbtext` and `$homecrumbiconid` properties. The first line in the data list is the first animated crumb to pop out from the Home crumb, with subsequent list lines being successive crumbs in the control.

The Breadcrumb control has the following properties:

<code>\$homecrumbiconid</code>	The icons id used for the home crumb
<code>\$homecrumbtext</code>	The text shown on the home crumb.
<code>\$crumbstyle</code>	The drawing style of the breadcrumb, a constant: <code>kCrumbStyleText</code> <code>kCrumbStyleRoundRect</code> <code>kCrumbStyleArrow</code>
<code>\$textcolor</code>	The crumb text color
<code>\$crumboutlinecolor</code>	The outline color of breadcrumbs
<code>\$crumbcolor</code>	The color of a breadcrumb
<code>\$activecrumbcolor</code>	The color of the active breadcrumb
<code>\$activecrumbtextcolor</code>	The text color of the active breadcrumb
<code>\$showactivecrumb</code>	If <code>kTrue</code> the active crumb (the final crumb on the path) is shown in the active color

Events

The Breadcrumb control reports the `evBreadcrumbPathChanging` event which is sent when the user selected a crumb in the path, with the event parameters `pBreadcrumbID`, which is the id of the crumb (column 2 in the crumb list definition), and `pNewBreadcrumbList` the proposed new path list.

This event can be discarded with `Quit Event (Discard Event)` which will prevent the default action which is to shrink the path based on the crumb selected.

Example

The following code will create a breadcrumb as shown in the examples above.

```
# where bread_crumb is a list assigned to $dataname of the Breadcrumb control
Do bread_crumb.$define(crumb_text,crumb_id,crumb_icon_id)
Do bread_crumb.$add("Clothing",1,"")
Do bread_crumb.$add("T-Shirts",2,"")
Do bread_crumb.$add("Red-T-Shirts",3,"")
Redraw bread_crumb_control
```

There is an example app to demonstrate the Breadcrumb window control on the Omnis GitHub repo at: <https://github.com/OmnisStudio>. Search for Omnis-Breadcrumb.

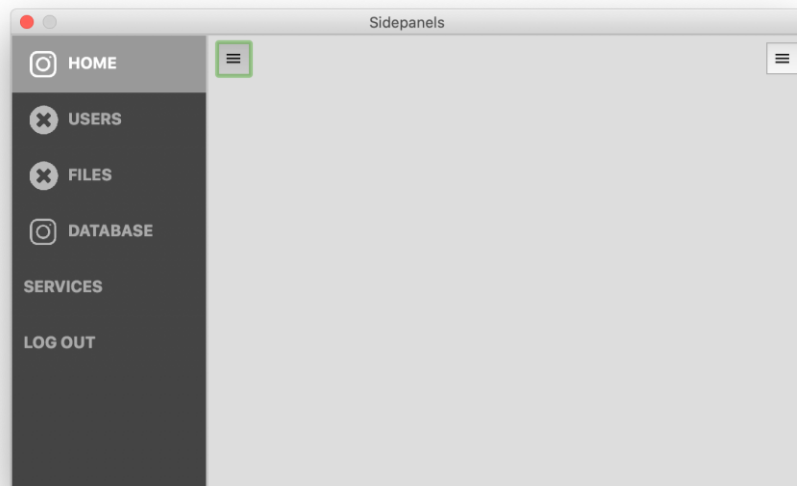
Side Panels

Side panels are a common UI element in many dashboard style designs seen today. A Side panel is a vertical panel down the side of a window, containing clickable options, such as a menu of options or other content, that can pop out on the left or right side of a window. A side panel can be shown automatically, when the user hovers the pointer over the left or right edge of the window, or linked to a button or menu option to allow the side panel to be opened or closed manually by the end user. When a side panel is opened it is animated.

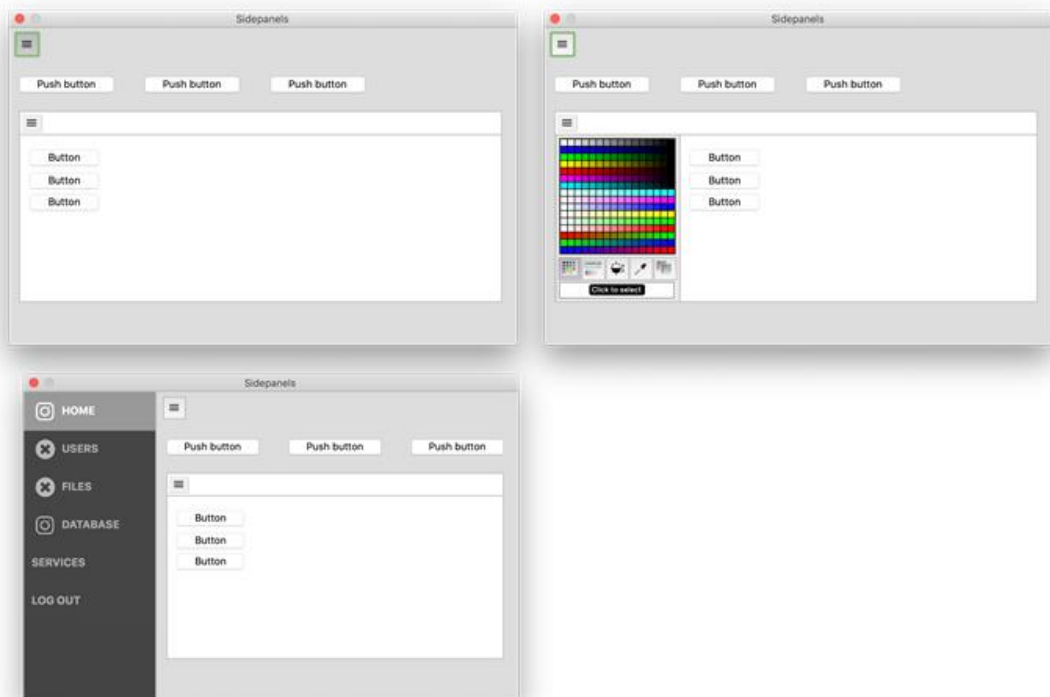
Note that Side Panels are available for window controls only, not JS client controls.

There is not a separate Side Panel component, rather a side panel is a property of a window component (`$sidepanel`, see below), so for example, you can create a Side panel using a Page pane. To create or enable a side panel, you need to set the `$edgefloat` property of a control to either `KEFposnLeftToolbar` or `KEFposnRightToolbar` and the `$sidepanel` property will become enabled. Any window object *that can be marked as a left or right toolbar* will have the `$sidepanel` property and therefore can be enabled as a side panel. However, from a practical point of view, it would normally make sense to use a container type field, such as a Page pane, Scroll box, or Group box as a side panel, since you can then add other controls to the container which the end user can interact with.

The following example window contains a Scroll box on the left, enabled as a side panel, which contains a vertical tab strip containing a number of clickable options; in this case, the small “hamburger” button can be used to hide or show the side panel.



You can also place one container type control inside another container and enable the second control as a side panel; for example, you could place a scroll box inside a page pane and make the scroll box a side panel. Using containers and other controls in this way, you can create some highly interactive interfaces or layouts in your application, such as the following:



Properties

When a window component is marked as a left or right toolbar (via `$edgefloat`), the **\$sidepanel** property is enabled, and once enabled, you can set the property to `kTrue` to enable the side panel behavior. You can then set the **\$sidepanelmode** property (the “peek” mode) so the side panel is shown and hidden automatically when the end user hovers their pointer over an area to the left or right of the window. The threshold for the peek area is 20 pixels on the left and right panels.

The `$sidepanelmode` property can be set to either a “push” or “cover” mode, as follows:

- ❑ **kSidePanelModePush**
this mode pops out the panel automatically and *pushes* or moves the other controls and content on the window either to the right or left.
- ❑ **kSidePanelModeCover**
this mode pops out the panel automatically which is placed *over the top* of the other controls and content on the window.
- ❑ **kSidePanelModeNone**
the default mode meaning the side panel will not pop out automatically, when the end user hovers over the window edge, but the `$showpanel` method can be used to show the side panel

When set to a mode, moving the mouse to where the side panel is fixed, either the left or right side of a window or a container, the panel will automatically animate or pop out. Moving the mouse out of the panel will autohide the panel.

When a side panel is closed (not visible), the panel is internally disabled for tabs. This prevents Omnis from tabbing to any controls within a hidden panel. When disabled, no events are sent.

Side panels support `$dragborder` meaning if the panel is opened and the border of the control is dragged, closing and re-opening the panel will open to the new dragged size.

Methods

Having enabled a control to behave as a side panel, by setting `$sidepanel` to `kTrue`, you can hide and show the panel manually, in your code, using the `$showpanel` method. In this case, you can set the `$sidepanelmode` property to `kSidePanelModeNone` and show or hide the panel using a button and this method.

- ❑ **`$showpanel(iAction, [iMode])`**
performs an action (`iAction`) such as `kSidePanelActionShow` on a side panel object. The panel mode (`iMode`) is optional, is the display mode (as above) such as `kSidePanelModeCover` and only used when showing a panel

The side panel action constants are:

- ❑ **`kSidePanelActionHide`**
hides the side panel
- ❑ **`kSidePanelActionShow`**
shows the side panel
- ❑ **`kSidePanelActionToggle`**
either hides or shows the side panel depending on its current state

For example, the following code for a button shows a scroll box name 'panel' that is enabled as a side panel:

```
On evClick  
  Do $cinst.$objs.panel.$showpanel(kSidePanelActionToggle,kSidePanelModePush)
```

Design Mode

In design mode, you can hide or show the side panel(s) using the window context menu, which would allow you to design the remainder of the window without the side panel getting in the way. When you Right-click on a window that has side panels enabled, the **Show Panels** submenu option allows you to hide or show the Left and/or Right side panels in design mode. If no panels are enabled for the window or container, the menu items are not present. In addition, clicking on a side panel in design mode will show a small semi-transparent button (shown below), which also allows you to hide or show the panel.



Example

There is an example app to demonstrate Side Panels on the Omnis GitHub repo at: <https://github.com/OmnisStudio>. Search for Omnis-SidePanels.

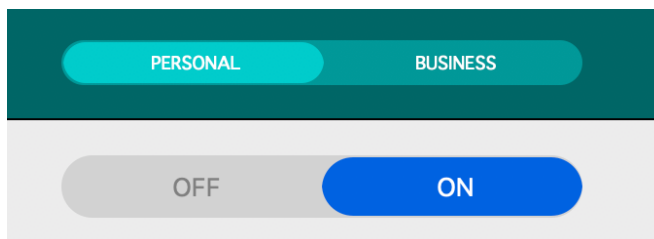
Check Box

The Check Box window class control has a new “horizontal” mode which makes the check box look and behave like a slider switch, which animates between the on/off state. The check box control has a new `$buttonmode`, `kCheckBoxHorizontal`, to enable the horizontal behavior.

The existing appearance properties `$textcolor`, `$forecolor` and `$iconid` specify the text color, forecolor, and icon for the selected state, and `$backcolor` is the background of the switch, while new properties `$secondaryforecolor`, `$secondarytextcolor`, and `$secondaryiconid` specify the appearance for the off state of the control.

The horizontal check box animates by default but this can be disabled by setting `$animateui` to `kFalse`. The animation is disabled when the horizontal check box is used in a complex grid.

When button mode is `kCheckBoxHorizontal` the `$text` property is a comma separated list allowing you to specify the text for the off and on states of the control. For example, `$text = “PERSONAL,BUSINESS”`, or `$text = “OFF,ON”` will display the check box as follows:



There are a number of new theme colors to control the appearance of the horizontal check box; by default the theme colors are set to `kColorDefault`.

<code>horizontalcheckbox</code>	the appearance group name for this control.
<code>background</code>	background color of the control.
<code>switchon</code>	background color of the switch in the ‘on’ state
<code>switchontextcolor</code>	text color of the switch in the ‘on’ state
<code>switchoff</code>	background color of the switch in the ‘off’ state
<code>switchofftextcolor</code>	text color of the switch in the ‘off’ state

There is an example app to demonstrate the Check Box window control, including the horizontal mode, on the Omnis GitHub repo at: <https://github.com/OmnisStudio>. Search for Omnis-CheckRadio.

Event & Control Methods

\$sendevent method

The \$sendevent method has been added to window objects and window instances to allow you to test your \$event/\$control methods for window fields or window instances (note this is for window fields and classes only, and not JS remote forms).

❑ \$sendevent(iEvent[,eventParameters...])

Sends event iEvent (an ev... constant value) to the object with eventParameters passed as name,value pairs, for example \$sendevent(evClick,'pLineNumber',2). Returns kFalse if the event is discarded; generates a debug error if there is a problem with the parameters.

You can also pass #SHIFT, #CTRL/#COMMAND, #ALT/#OPTION as "event parameter" names, in order to set the value of these variables when the event is being executed.

When entering a \$sendevent method, typing Ctrl+Space after the quote lists the event parameter names.

Note that the invoked \$event/\$control will execute, but if there are requirements of the data associated with the control, you need to separately code for that - \$sendevent simply sends the event causing \$event or \$control to execute appropriately.

Complex Grid

Sliding columns

Complex grids can now have Left or Right sliding columns. There is a new property \$haslideoutcolumn which controls the sliding columns on the complex grid, taking one of the following constants: kTableColumnsNormal, kTableColumnsLeftSlide, kTableColumnsRightSlide, or kTableColumnsLeftRightSlide.

There is a new method \$slideoutcolumn(kTableSlideDirection..., iRow=-1, kTableColumn...) which sets iRow. If iRow=-1 the current row is hidden/shown/toggled, while kTableColumn... controls if the row's left or right column slides out. The kTableSlideDirection... constants are: kTableSlideDirectionToggle, kTableSlideDirectionHide, or kTableSlideDirectionShow.

There is an example app to demonstrate Sliding Columns in complex grids on the Omnis GitHub repo at: <https://github.com/OmnisStudio>. Search for Omnis-ComplexGridSlide.

Grid Exceptions

\$width and \$height are now supported as table exception properties for objects referenced via \$obj and \$bobj, however, support for exceptions to objects in \$obj is limited to non-enterable tables only.

Other Grid enhancements

Complex grids now support mouseover(kMHorzCell) returning the column clicked allowing you to detect clicks in slide out columns.

Complex grids now support the \$hiliteLine and \$dropbetweenlines properties.

List Control

Queue click: pLineNumber

The pLineNumber event parameter is now set for evClick and evDoubleClick events when generated via *Queue click* or *Queue double click* to a list or list sub-class.

OBrowser

Setting Locale

The OBrowser control now offers more control over the client's locale. A new "locale" attribute has been added to the "obrowser" section in config.json, for example:

```
"obrowser": {
    "locale": "it_IT"
}
```

The locale attribute allows you to specify a language to be used in OBrowser (e.g. in context menu entries), and it can also be used to change the Accept-Language header, which servers can take into account to serve a language-specific page. For example, if the locale is set in config.json to it_IT and the \$urlcontrolname property is http://www.google.com, Omnis will load Google's search engine page in Italian. When locale is not specified in config.json, a locale of en-GB is used by default. Note that the Accept-Language is only "for information" for the server, therefore, the server can send back a web page in English even if you requested another language.

In addition, there is a new property in OBrowser, \$acceptlanguagelist, which can be set to a comma-delimited ordered list of language codes (ISO-639, without any white space), that is used in the Accept-Language HTTP header. For example, it-IT will try loading a web page in Italian, however the server determines whether the web page it sends back is in the requested language in Accept-Language or not. If \$acceptlanguagelist is not specified (the default), then the value of the locale attribute in "obrowser" in the config.json is used. If specified, it will override the value in the attribute.

It is important to note that the \$acceptlanguagelist will take effect only upon OBrowser component initialization. This means that you will need to set \$acceptlanguagelist, close the window containing OBrowser in order to destroy the OBrowser instance and re-open the window so OBrowser re-initializes and uses the recently set \$acceptlanguagelist values.

Port setting for 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.

Certificate errors

A new property \$ignorecertificateerrors has been added to OBrowser to handle errors when the component encounters errors with the website certificate.

If true, a URL with certificate errors will be allowed to be loaded. If false (the default), an error will be raised and sent via the evBrowserFrameLoadError event. When an error is raised, the message in pErrorText will be "ERR_CERT:The certificate for this server is invalid:<errorcode>". This message can be tested to provide a prompt to the user to allow them to proceed to the insecure page after setting this property to true.

Edit Controls

Dictation (macOS)

Support for Dictation is now turned on in Omnis by default (it was off by default in previous versions). To disable Dictation you need to edit the Omnis configuration file (config.json). There is a "useDictation" option in the "macOS" group in config.json, which must be set to false; note you have to quit Omnis in order to apply the change to the config.json file. Dictation will be disabled when you restart Omnis.

```
"macOS": {
```

```
"useDictation": false
}
```

Dictation Level and Voice Control

Enhanced dictation has been replaced with Voice Control in macOS Catalina and later.

Voice Control has been introduced in macOS Catalina to improve on and replace the earlier Enhanced Dictation feature. Speech can be dictated in Studio via Voice Control when Dictation is enabled in Studio. Voice Control is enabled via the Accessibility System Preference. When an edit field in Studio is accepting input and Voice Control is active then speech input will be translated into text via the Voice Control speech engine.

Tooltips

Tooltips for Window class components have been improved whereby you can control the background and text colors, the justification of text and the position of the tooltip relative to the component. In addition, tooltips now fade in and fade out, although the latter does not apply when immediately replacing a tooltip with new content.

There are some new properties in the tooltip section of appearance.json that control the appearance of tooltips.

- systemstyle**
If true (the default), tooltips in the system style are drawn using `colorinfobk` and `colorinfotext`; if false, Omnis style tooltips are drawn using `tooltipbackgroundcolor` and `tooltiptextcolor`
Omnis style tooltips have rounded corners and (unless it is not relevant for the particular tooltip) a small pointer
- tooltipbackgroundcolor**
The background color used for Omnis style tooltips
- tooltiptextcolor**
The text color used for Omnis style tooltips
- defaultjustification**
The default justification of the tooltip rectangle relative to the active area of the component: 0 left (the default), 1 right, 2 center; ignored by certain controls if applying the default does not make sense

In addition, window components have a new property:

- \$tooltippos**
A `kTooltipPos...` constant that specifies where `$tooltip` appears relative to the window control

<code>kTooltipPosMouse</code>	<code>\$tooltip</code> appears relative to the mouse pointer (this is the default and corresponds to 10.1 behavior)
<code>kTooltipPosRight</code>	<code>\$tooltip</code> appears to the right of the window object
<code>kTooltipPosBottom</code>	<code>\$tooltip</code> appears below the window object
<code>kTooltipPosLeft</code>	<code>\$tooltip</code> appears to the left of the window object
<code>kTooltipPosTop</code>	<code>\$tooltip</code> appears above the window object

The enhancements to tooltips also applies to tooltips that appear in the Omnis IDE including the Studio Browser and Code Editor. Specifically, the Code Assistant parameter help popup is displayed using the new style tooltip.

Dropdown Lists

The color and fill pattern of the Dropdown List window class component on Windows now respects the `$backgroundtheme` property and only defaults to Windows system

colors when the theme is set to `kBGThemeControl`. For other background themes it uses the appropriate colors and fill pattern, and for `kBGThemeNone`, it uses the foreground and background colors and fill pattern.

Window Fields

There is a new property to show the contents of disabled window fields as greyed out. The Entry, Masked Entry, and Multiline Entry window field types (`kEntry`, `kMaskedEntry`, `kMultilineEntry`) now have the property, `$fadewhendisabled`. When `kTrue` (the default is `kFalse`), and if the field's `$enabled` property is `kFalse`, the field content will be partially greyed out.

There are two new theme colors in `appearance.json` (and the theme templates) to control the color and transparency of `$fadewhendisabled`: "fadewhendisabledcolor" (defaults to `kColorWindow`) and "fadewhendisabledalpha" is the amount of alpha used when fading to `kColorWindow` (default is 140).

Tab Strip

Object animation was added for various window class components in Studio 10.1 including the Tab strip control: this has been enhanced in Studio 10.2 by adding a vertical mode, with the new property `$verticaltabs`, for all the new animated tab strip styles.

The tabs can be set using `$tabs`, but now tabs in the animated styles have extra options for the icon and caption: `$tabiconid` and `$tabcaption` can be set under the 'Pane' tab in the Property Manager (like Page pane and Tab pane).

When the tab strip has the focus, the Left, Right, Up, and Down keys can be used to change the current tab.

There is an example app to demonstrate the Tabstrip animation on the Omnis GitHub repo at: <https://github.com/OmnisStudio>. Search for Omnis-TabStrip.

Headed Lists

In order to display a 16x16 icon in the header in a Headed List, you need to set a minimum column width of 21 pixels, to allow for the width of the icon and the padding Omnis adds to a column.

HTML Controls

The `omn_list_base.js` and `omn_date.js` files have been bundled with `omnishtmlcontrol.js` so you only need to reference the one file, `omnishtmlcontrol.js`. Existing HTML controls should work as the bundle is named `omnishtmlcontrol.js`, which is also now minified.

Focus Field Style

Window objects have a new property `$fieldstylefocused` to allow you to style an object when it gets the focus, allowing you to override certain properties when a field has the focus such as the font style or color.

Picture fields

Image Interpolation

By default, Omnis interpolates (smooths) an image when rendering it on ultra-high definition displays. There is a new property for Picture fields, `$nointerpolation` (default is `kFalse`), which allows you to disable interpolation, which may not be required for certain types of image, for example, for displaying a bar code (macOS only).

Moving Objects into Group boxes

The `$objlink` property is an integer that contains information about the container of the object. You can now assign `$objlink` in your code using class notation, provided that *the*

design window is not open. So for example, you can now move an existing control in a window class into a Group Box in the same window class using code.

Checked Menu Items

In previous versions Studio 10.0 and 10.1, Menu items with a single-state icon did not show the checked state on Windows. These menu items now draw on a rectangular colored background, using the same color as checked menu items which do not have an icon.

Toolbar Spacers (Windows)

Toolbar spacers are now displayed correctly on Windows (version 8 and 10). Existing toolbar spacer objects need to have their \$blank property set to kTrue to restore their previous (blank) appearance on Windows.

Tray Icon (Windows)

The mouse coordinates passed to the menu event are now relative to the main Omnis Window; these now work correctly with the PopupMenu command, for example. When Omnis is minimized, these coordinates will be larger than expected (due to how Windows 10 handles them), but they can still be used with PopupMenu to open the menu in the correct position.

Window Programming

The following new features and enhancements are for Window Classes (not JavaScript remote forms).

Example Apps

A number of sample apps demonstrating new Thick Client controls or enhancements have been placed on the Omnis GitHub repo at: <https://github.com/OmnisStudio>. These include: Toast Messages (Omnis-Toasts), and System Drag and Drop (Omnis-SystemDrop).

Toast Messages

Toast messages are small notification type messages that that can be “popped up” in your application to alert the end user about something; toast messages were previously available for remote forms (using a client command), but this enhancement allows you to open toast messages in your desktop apps, via a window instance (\$cinst) using a new \$showtoast method.

Toast messages have a title, message and an icon, and can be positioned in the top-left, top-right, bottom-left, or bottom-right relative to the Omnis application window (not the desktop) under Windows, or the whole screen on macOS. They will close automatically after 4000 ms or a specified time.

Toast messages are non modal, and therefore they are outside the scope of other Omnis window stacks and do not interfere with evToTop message processing, nor do they change the end users current window or current focused object.

They have the following UI layout:



The following are some example toast messages:



Toast messages have close boxes so they can be dismissed before the default time expires. They have a set of predefined colors and default width with the option to override these colors by setting some new appearance theme settings. The message box has a progress timer bar to show how long remains before the message box is dismissed.

Toast messages can be stacked up to 6 levels deep. When a message is manually closed or expires, it is removed from the stack and other on-screen messages adjust keeping a stacked appearance.



\$showtoast method

Toast messages are opened in the context of the current instance (window or object instance) using the **\$showtoast()** method.

- ❑ `$cinst.$showtoast(cTitle, cMessage, iStyle, [iStack, iTimer, bClearStack])` adds a new toast to a stack
 - cTitle: Text or empty "
 - cMessage: Message text
 - iStyle: kToastSuccess, kToastError, kToastWarning, kToastInformation
 - iStack: kToastTopLeft, kToastTopRight (the default), kToastBottomLeft, kToastBottomRight
 - iTimer: number of milliseconds the toast will be displayed (default is 4000, as set in theme)
 - bClearStack: if kFalse (the default) new toast messages are stacked with previous messages, otherwise if passed as kTrue, all previous toasts in stack will be cleared

For example:

```
$cinst.$showtoast('Success','Congratulations your order is complete',
  kToastSuccess )
$cinst.$showtoast('Error','Problems with your connection.', kToastError,
  kToastTopLeft )
```

Note: Attempting to add a toast to the same stack with an identical title and message will reset the existing toast timer and not add a new toast.

Toast Message Colors

You can override the default colors of the four toast styles, the onscreen delay, and the default width in the 'toast' section of the appearance.json file.

```
toast
{
  toastsuccessbackgroundcolor : color
  toastsuccesscolor : color
  toastsuccessicon : number

  toasterrorbackgroundcolor : color
  toasterrorcolor : color
  toasterroricon : number

  toastwarningbackgroundcolor : color
  toastwarningcolor : color
  toastwarningicon : number

  toastinformationbackgroundcolor : color
  toastinformationcolor : color
  toastinformationicon : number

  toastdefaultdelay: 4000
  toastdefaultwidth: 400
}
```

The following example code opens different toast messages:

```
If (toastType>kToastInformation)
  Calculate toastType as kToastSuccess
End If
Calculate type as toastType
Switch type
  Case kToastSuccess
    Calculate title as "Success"
    Calculate message as "Congratulations. Logon complete."
  Case kToastError
    Calculate title as "Error"
    Calculate message as "Sorry your details are incorrect. Please
    check your deails and try agin."
  Case kToastWarning
    Calculate title as "Warning"
    Calculate message as "Please check your settings"
  Case kToastInformation
    Calculate title as "Information"
    Calculate message as con('Great news. All formst have been
    submitted, saved and has passed ', style(kEscBmp,1613), ' You can
    now move on to the next level. ')
End Switch
Calculate toastType as toastType+1
```

There is an example app on the Omnis GitHub repo at: <https://github.com/OmnisStudio> to demonstrate how you can implement Toast Messages. Search for Omnis-Toasts.

Drag and Drop: System Files

Support for dragging and dropping *operating system files* and *file data* (in the thick client) has been simplified providing more control in your event handling code. As a

consequence there may be some minor compatibility issues, but these are outlined below.

There is an example app to demonstrate system file drag and drop on the Omnis GitHub repo at: <https://github.com/OmnisStudio>. Search for Omnis-SystemDrop.

Drop mode

The settings for \$dropmode have changed: kAcceptFiles and kAcceptFileData have been replaced with a single mode, kAcceptOperatingSystem: specifically, kAcceptFiles has been renamed kAcceptFiles_OBSOLETE, and kAcceptFileData has become kAcceptOperatingSystem. These changes are made automatically on conversion and both of the old modes result in the kAcceptOperatingSystem mode; whether data can be dropped is handled differently, as described below.

Therefore, to accept dropped files or file data from outside Omnis, you must now set \$dropmode to kAcceptOperatingSystem. Note that as in previous versions, kAcceptAll does not select kAcceptOperatingSystem; kAcceptAll means accept drops from all types of Omnis control.

Drop mode flags

Window controls that have the \$dropmode property (as well as the thick client window itself) have a new \$osdropflags property to control what is dropped (data or files). This is the combination of a number of new constants that can be used to specify what is dropped:

- kOSDROPincludeData**
If set in \$osdropflags, pDragValue will include the data for objects dropped from the operating system if the data is available and kOSDROPwithoutDataIfOsDropLimitExceeded allows.
- kOSDROPfilesOnly**
If set in \$osdropflags, objects dropped from the operating system must all be files. Note that on macOS, data provided by file promises may not be accepted, because the file containing the data may only become available when dropping the object(s).
- kOSDROPwithoutDataIfOsDropLimitExceeded**
If set in \$osdropflags, \$clib.\$prefs.\$osdroplimit can be exceeded (in which case data is not included in pDragValue for evDrop)

On conversion, objects that were set to kAcceptFileData are now set to kOSDROPincludeData.

Drop data limit

A new library preference \$osdroplimit (\$clib.\$prefs) sets the maximum number of bytes of dropped data that can be included in pDragValue for evDrop when \$osdropflags contains the flag kOSDROPincludeData; it defaults to 100000000 (100MB). The setting kOSDROPwithoutDataIfOsDropLimitExceeded specifies if evDrop still occurs when the limit is exceeded.

Note that combining kOSDROPincludeData and kOSDROPwithoutDataIfOsDropLimitExceeded with a suitable drop limit provides a good way of accepting files of arbitrary size (where the data is too large to be read into the drag value) and also accepting other non-file objects.

Note that if you change \$osdroplimit, any windows relying on its value need to be closed and re-opened.

Event Parameters

There have been some changes with the parameters for the evDrop event: pDragType was previously set to kDragFiles but is now set to kDragOperatingSystem. While kDragFileData is renamed to kDragOperatingSystemData_OBSOLETE, and resolves to the same value as kDragOperatingSystem.

The `pDragValue` list uses the same column names as previous versions (for compatibility), and has an additional column called **isfile**, a Boolean, which is true if the dropped object is a file.

Notes that **filedata** is now always a binary value containing the contents of the object, and **filesize** is a 64 bit integer.

When dropping email from macOS Mail, `pDragValue` has the same content (with these additional columns) as previous versions.

File extension (macOS)

For mac OS, when a file is dragged a new fourth column `pFileextmac` has been added to the `pDragValue` parameter which returns the macOS file extension, or is empty if the data is from an unknown application.

Notes for existing users:

Importing classes from JSON applies appropriate conversions to constant names and `$dropmode`.

The old-style Windows file drag and drop, enabled using "classicwindowssystemdragdrop" in the "windows" section of `config.json` is no longer supported.

The Catalog String Table tab, and the runtime data file browser, have been changed to work with the new drag and drop implementation.

Menu and Toolbar Fonts (Windows)

You can now scale menu and toolbar fonts when using design DPI scaling. Specifically, window menus and window toolbars, under Windows, scale using the design DPI preferences and settings for the library.

There are some new entries in the **config.json** file that can be used to control scaling of menus and toolbars for cases where Omnis cannot easily determine a library to be used as the source of the scaling settings. These entries are:

- defaultMenuDesignDPIMode** and **defaultMenuDesignDPI**
in the "windows" section (these only apply on the Windows platform: note that menu items are never scaled on macOS)
- windowToolbarDesignDPIMode** and **windowToolbarDesignDPI**
in the "ide" section
- dockingAreaDesignDPIMode** and **dockingAreaDesignDPI**
in the "ide" section

The syntax for these entries is

- the mode entries: "kDPIall", "kDPIoff" or "kDPIframeOnly"
- the DPI values: 3 comma separated DPI values

This corresponds to the syntax of the library preference (even in the case of menus where only the Windows platform value is used). Typically, you would set these to the same value as those in the library preferences.

Omnis Libraries

Starting Omnis

There is more control or reporting over when Omnis is started via a file, such as a library file (or Omnis data file).

The **sys(250)** function has been added which returns a list of files which were used to open Omnis, e.g. double-clicked from the Finder or passed on the command line. This is empty if Omnis was opened directly by double-clicking.

In addition, a new task method called \$openfiles has been added. This method can be overridden and will be called when Omnis is used to open a file or set of files by the OS. This will be passed the list of files as a parameter.

On Windows, the \$singleinstance root preference needs to be set to kTrue to use the same instance of Omnis to open a file, otherwise another instance of Omnis will be started.

You can now open a library in the system File Explorer in Windows or the Finder on macOS from the context menu for a library when listed in the Studio Browser.

Opening Initial File As Library error

The reportErrorOpeningInitialFileAsLibrary option has been added to the “defaults” section of the config.json file (default value true) to control whether or not Omnis reports an error trying to open the initial file as a library; this applies when a file is dropped on the Omnis program, or the file is double-clicked. If the option is set to false the error message is not shown.

Library Parent folder

The \$parentfolder property has been added to a library and it returns the pathname of the folder containing the library file (with a trailing pathname separator). Note that this is only visible in the Property Manager when the properties of the library are accessed via the Notation Inspector. Using this new property you could, for example, find the full notation for the library folder (path) and drag it to the Code Editor.

Export to JSON & LFs

When exporting a library to JSON, LF (linefeed) characters in code are now exported as Unicode private-use character 0x2fffe, to reduce issues with other tools (note CR characters are also mapped to 0x2ffff).

Omnis Environment

The following new features and enhancements are related to the Omnis IDE and various tools.

Regular Expressions

The way you can construct regular expressions in Omnis in your code has been enhanced, with the addition of the PCRE2 library (*Perl Compatible Regular Expressions* version 2). PCRE2 is an open source library of functions that provides syntax and semantics like Perl 5 for defining a search (see www.pcre.org for more information and full documentation).

Everywhere you can use a regular expression in Omnis you can now use a PCRE2 compatible regular expression. If you want to use the old regular expression syntax, you can set the `useOldRegularExpressionSyntax` configuration option to true (false is the default, so PCRE2 is used by default); this is in the 'defaults' section of the `config.json` file. When this is set to true, it only affects the `rxpos()` function.

PCRE2 provides improved error message reporting when there are problems with regular expression syntax, and these are now reported where applicable. For example, the command filter and minimum command length files used in the Code Editor can contain regular expressions, so errors with these are written to the trace log at startup.

An error with the regular expression passed to the `rxpos()` function generates a debugger error with the specific error text rather than a generic invalid regular expression error.

However, when using the find field in the Code Editor, note that errors are not reported because the editor attempts to compile and use the regular expression on every keystroke.

With the introduction of PCRE2 the `rxpos()` function has been enhanced; see the functions section.

Auto Save

There is a new **Auto Save** option in the File menu that when enabled means that Omnis will save all classes that are currently open in design mode automatically; the option defaults to disabled, meaning that you have to save a class manually or the class is saved when it is closed (as in previous versions).

The state of the Auto Save option is saved under a new "autoSave" option in "ide" section of the `config.json` configuration file. The interval between each auto save can be configured in a new "autoSaveInterval" option, also in the "ide" section of `config.json`: this is the number of milliseconds between each auto save, which is set to 1000 by default.

The **Revert** menu and toolbar command is not available when Auto Save is enabled.

Auto Save applies to all class and method editors except for system classes, provided that the class is not read-only, and the method editor is not in read-only mode.

Search Catalog and Interface Manager

A new search box has been added to the **Catalog** and **Interface Manager** to allow you to search for items, or to filter the list of items displayed, to help you to find items more easily; this is in addition to the search box that was added to the Property Manager in a previous version, plus there is a search box in the new Select Icon dialog.

For the Catalog, Interface Manager, and Property Manager, the **Clear Search** button (x) is now to the left of the search box. In addition, the clear search button is now always visible, and set to read-only when there is no search text to clear.

Note that the 'propertyManagerSearch' entry in keys.json (introduced in a previous version) has been renamed 'search', and now applies to the Catalog and Interface Manager search as well as the existing search box on the Property Manager.

The search box on the **Catalog** allows you to search for classes, functions, constants and other items that appear on separate tabs in the Catalog. The search results are all matching items (still in their groups), together with groups that have no match where the group name matches. Using search next/previous in the Catalog cycles through all groups identified by the search that contain at least one matching item.

The search box on the **Interface Manager** replaces the toolbar (which just had a simple view menu, which is now accessible only via a context menu). The search results are all matching methods and properties (still associated with their class or field), together with any names in the field name list which match the search string.

For both the Catalog and Interface Manager the search function allows you to cycle through the results using some new keys (in the IDE section of keys.json), **searchNext** and **searchPrev** which are mapped to Ctrl+G and Ctrl+Shift+G, respectively.

Using search next/previous in the Catalog cycles through all tabs and fields identified by the search that contain at least one matching item. So, for example, if field *test* has matches in both methods and properties, ctrl+G will go to the methods first, and the next ctrl+G will go to the properties. If a control has no methods, but it does have properties, ctrl+G will go directly to the properties tab, and vice versa.

Property Manager

Showing all properties on one tab

You can display all the properties for a class or object on a single tab in the Property Manager by entering * in the Property Manager Search box (in effect, this matches and displays *all properties*).

Background color

The background color of the Property Manager is now set to a specific color rather than kColorDefault. The setting has been changed in the default and blue themes, as well as the templates.

Object Width and Height

The width and height (w x h) of the area occupied by a group of selected remote form or window class objects is now shown on the status bar of the Property Manager.

Find and Replace

Finding Folder

You can now select classes to search using Find & Replace based on the name of their parent folder. There is a new button in the title of the class list (with a folder icon), in the Find & Replace window, that allows you to search for parent folder(s) using a regular expression, and then select the classes contained in those folders in the class list.

There are new keyboard shortcuts in a new findAndReplace group in keys.json that operate the three buttons in the class list title. In addition, you can now tab to the tabbed pane (to operate it with the keyboard), and on Windows there are keyboard accelerators for the buttons.

Selected Class

If the Find & Replace dialog is already open and you bring it to the top, it will now select the top-most open class.

Search Selected Methods

The "Use selected method" and "Only lines containing selection" options have been added to Find and Replace to help you find items while working in the Code Editor (existing users will note these options were in versions up to Studio 8.1).

To use these new options you need to select at least one character in the method: Find and Replace searches all lines containing a part of the selection, and when it completes, it selects the text for the searched (and possibly replaced) lines.

Code Syntax Colors in Find Log

The code syntax colors used in the Code Editor are now used to display method lines in the Find and Replace log, and the Trace log. You can set this using two new entries in the ide section of config.json. `findAndReplaceLogUsesSyntaxColors` and `traceLogUsesSyntaxColors`, which are both enabled by default.

Configuration File

Errors in config.json

Errors in the Omnis configuration file config.json are now written to the trace log. As Omnis is loaded, it parses the config.json file and if it fails, an error is written to the trace log.

Dock Key Events (macOS)

The `monitorDockKeyEvents` option has been added to "macOS" section of the config.json file to disable the Keystroke Receiving dialog at startup on macOS. If set to false, Omnis does not attempt to monitor keyboard events from the Dock and the dialog will not be shown. The option is set to true by default for backwards compatibility.

Save Window Setup Shortcut

A new keyboard shortcut Shift+F3 has been added to execute the Save Window Setup command for the current design window; the new shortcut applies to all built-in dialogs and design windows. A new option 'saveWindowSetup' has been added to the IDE section of the keys.json file to store the shortcut.

Function key shortcuts in macOS menus are shown as Fn rather than Cmd+<n>.

View Menu

Recent Classes

The **View** menu lists all the classes or methods that have been opened recently. The maximum number of classes shown in previous versions was limited to 9, but now you can configure the number of classes shown. There is a new entry in config.json, called "maxRecentClassEntries" in the "ide" section, which defaults to 9 (the value in earlier versions), but can be set to any value in the range 9 to 32 inclusive.

Note that this also affects the **Recent Classes** hyperlink in the Studio Browser, but since that only shows classes (or methods when the Shift key is pressed), there are typically fewer recent class items on the recent classes hyperlink than on the main View menu.

SQL Query Builder

You can now change the max line height in the result window in the Query Builder and Interactive SQL windows using the context menu on the results grid; the setting is saved in the Save Window Setup.

Localization

Localization Optimization

Localization of the built-in strings that may appear in the JavaScript Client has been optimized to make it easier for developers, as well as reducing data size for applications that support multiple languages by only loading the required language file(s) if they are needed. The JS client now supports German, French, Italian and Spanish translations by default, but support for other languages can be added as required.

Localized String files

There is a new folder in the Omnis tree under `html/scripts/` called 'locale'. This stores .js files which contain translated strings, using either a 2 or 4 letter language code, and must be named in the format `strings_[language code].js`; for example, `strings_en.js` for standard English, or `strings_en_us.js` for American English. Inside each file is an object, which is a member of `jOmnisStrings`, containing key-value pairs to translate to the given language. The member name should match the language code given in the file name, therefore, for french, the `strings_fr.js` file contains an object `jOmnisStrings.fr`. The `strings_base.js` file contains the base strings, and should always be present. There is a template file called `strings_template.js`, which provides more information about creating your own translations, with comments for each key-value to help you understand what each string is used for.

Setting the supported languages

In addition, there is a new global variable, 'supportedLanguages', which is defined in the `htm` file for a remote form (i.e. the application). This contains an array of language codes for the supported languages. On loading a remote form, the locale of the client is detected, and then checked against the supported languages in the array. It will look for both the 4 and 2 letter language codes, and if found, will request `strings_base.js` and the relevant localized versions. This means the JS client will only download the strings it needs.

Compatibility

Applications that use the existing localization will not be broken by the new localization support, but it is recommended that you switch to using the new mechanism to benefit from the performance improvements.

Managing the Client Locale

Two new *client commands* have been added to allow you to override the locale on the client device in the JavaScript Client.

- ❑ **"setlocale"** takes a row with 2 parameters
 - `cLocale` - a string containing valid locale code (e.g. "fr", "en", etc)
 - `[cPromptToReload]` - defaults to false (no dialog opens), if true pops up a no/yes dialog asking the user if they wish to reload the page for language changes to take affect. Defaults to No to reduce the chance of the user accidentally selecting yes and losing any unsaved data.
- ❑ **"clearlocale"** takes a row with 1 optional parameter
 - `[cPromptToReload]` - as above

The application needs to be restarted on the client for the change in locale to have an effect. Note that the client commands set this as a `localStorage` preference, so all Omnis JS client applications (forms) on the same client device will use this setting.

Report Programming

Screen Destination

The **Screen** print destination is now obsolete and is mapped to the Preview destination by default. There is a new option "useScreenDestination" in the "defaults" section of config.json and when set to true allows a report to be sent to the screen.

The *Send to screen* command is now obsolete and is therefore no longer shown in the Code Editor, but will continue to work in converted libraries. In addition, the kDevScreen destination is also hidden in the Code Assistant and in the Notation Inspector list of devices.

Report Page Preview

The Page preview window can now be opened maximized by specifying /MAX in the *Send to page preview* command parameters or in \$windowprefs.

Screen Report Fields

The \$pagecount and \$currentpage properties have been added to screen report fields. The current page count \$pagecount is read only, while \$currentpage is the currently displayed page and is assignable at runtime. When more than one page is visible, the value indicates the page that is most visible.

Print preferences

\$macosprintstatus

The Omnis preference \$macosprintstatus (\$root.\$prefs) has been removed, since it had no effect in more recent versions of Omnis Studio.

OW3 Worker Objects

There are some enhancements to the IMAP, HASH, and the FTP workers, plus support for OAUTH2 has been added to the HTTP, IMAP, POP3, and SMTP workers with the introduction of a new OAUTH2 Worker object.

OAUTH2 Worker Object

Support for OAUTH2 authorization has been added to Studio 10.2 by adding a new OAUTH2 Worker Object in the OW3 Worker component package, while the HTTP, IMAP, POP3, and SMTP workers have been modified to support OAUTH2 authorization via the new dedicated worker object.

Why use OAUTH2

OAUTH 2.0 provides much improved security over and above simple username and password schemes. It is commonly used by many different service providers, such as Google mail, for which its use will become mandatory in 2020 (meaning less secure apps will no longer be supported). You can read about OAUTH 2.0 in RFC 6749 (<https://tools.ietf.org/html/rfc6749>)

An application wishing to use a service (using HTTP, IMAP, POP3, or SMTP) requires an **Access Token** of type "bearer". The application needs to be registered with the service, so it can identify itself to the service, and the registration process provides the application with a Client Id and a Client Secret, that identify the application to the service.

As an initial step, the user of the service must authorize the application to use the service. To do this:

- ❑ The application opens a Web browser at the Authorization Endpoint (a URL) of the service.
- ❑ The authenticated user agrees that the application can access the service.
- ❑ The server hosting the Authorization URL redirects the browser to a URL supplied when opening the Web browser. This request contains an Authorization Code.
- ❑ The application makes a request to the Token Endpoint (a URL) sending it the Authorization Code.
- ❑ The server hosting the Token URL returns various pieces of information to the application, including: Access Token, Expiry of Access Token (recommended but not mandatory), Refresh Token (optional).

At this point, the application can use the Access Token to make requests to the service. Access Tokens *are short-lived, typically being valid for about an hour*. If the Token URL server also returned a Refresh Token, the application can use that after the Access Token has expired to obtain a new Access Token, without any further interaction with the user. Refresh Tokens typically have a long lifetime, but may be invalidated for various reasons, depending on the service implementation.

Obtaining Authorization

The new OAUTH2 Worker allows you to obtain an Authorization Code, exchange it for tokens, and refresh tokens, using the `$authorize()` method on a background thread. The worker also contains methods to save and load the tokens and other related information to and from an encrypted binary block of data, which helps to protect key pieces of information such as the Refresh Token and Client Secret.

Note that you must always use an Object Reference to create the OAUTH2Worker object – this eliminates potential issues with the way Omnis uses the OAUTH2Worker as a property value.

The object reference to an OAUTH2Worker object containing the authorization information can be passed to a new `$oauth2` property in the HTTP, IMAP, POP3, and SMTP Workers to provide authorization.

OAUTH2 Properties

These properties are specific to OAUTH2.

Property	Description
\$accesstoken	The access token to be used with HTTP, IMAP, POP3 and SMTP connections.
\$accesstokenexpiry	The expiry date and time of the access token (in UTC time). #NULL means no access expiry date and time is available.
\$authorizeurl	The URL of the OAUTH2 authorization endpoint.
\$clientid	The Client Id used in conjunction with the client secret to identify the application to the OAUTH2 authorization server.
\$clientsecret	The Client Secret used in conjunction with the Client Id to identify the application to the OAUTH2 authorization server.
\$redirecturiserveraddress	If not empty (the default value), this property overrides localhost in the redirect URI server address, replacing localhost with the value of this property. The default is localhost rather than 127.0.0.1 when generating redirect URIs when running in a thick client remote task
\$refresh token	The Refresh Token to be used to request a new access token after the access token has expired.
\$scope	A string identifying the type of access required. Used as part of the URL used to open the Web Browser at the authorization endpoint. For example, when using Google to access GMAIL, specify the scope as "https://mail.google.com/".
\$tokenurl	The URL of the OAUTH2 token endpoint.

HTTP and General Properties

In addition to the OAUTH2 properties, the OAUTH2Worker also has various HTTP and general OW3 properties, that for example affect how the HTTP requests it makes are executed: the OAUTH2Worker makes two different HTTP requests: a request to exchange an Authorization Code for an Access Token, and a request to obtain a new Access Token using the Refresh Token.

Property	Description
\$errorcode	Error code associated with the last action (zero means no error).
\$errortext	Error text associated with the last action (empty means no error).
\$followredirects	If true, the HTTP request will follow a server redirect in order to complete the request. Defaults to false
\$proxyserver	The URI of the proxy server to use for all requests from this object e.g. http://www.myproxy.com:8080. Must be set before executing \$run or \$start. Defaults to empty (no proxy server).
\$proxytunnel	If true, and \$proxyserver is not empty, requests are tunnelled through the HTTP proxy
\$proxyauthtype	The type of HTTP authentication to use when connecting to \$proxyserver. A kOW3httpAuthType... constant (see below).
\$proxyauthusername	The username used to authenticate the user when connecting

	to \$proxyserver using \$proxyauthtype.
\$proxyauthpassword	The password used to authenticate the user when connecting to \$proxyserver using \$proxyauthtype.
\$state	A kWorkerState... constant that indicates the current state of the worker object.
\$threadcount	The number of active background threads for all instances of this type of worker object.
\$timeout	The timeout (in seconds) for requests. Defaults to 60 with a minimum value of 10.
\$protocollog	If non-zero, the worker adds a log of protocol activity as a column named log to its wResults row. The desired value must be set before calling \$run or \$start. Defaults to kOW3logNone. Otherwise, a sum of kOW3log... constants.

OAuth2 Standard Methods

\$authorize

`$authorize([iAuthFlow=kOW3OAUTH2authFlowCodeWithPKCE])`

Starts the OAuth2 authorization flow `iAuthFlow` on a background thread. Returns true if the thread was successfully started. Properties of the object cannot be assigned while `$authorize()` is running.

`$authorize()` opens a Web Browser at the authorization URL, passing the URL various parameters in the query string, such as the Client Id using the value of the `$clientid` property.

How the Web Browser is opened depends on the context in which `$authorize()` is called.

When executed within the context of a **thick client** (non-remote) task, `$authorize()` uses the `$webbrowser` property to control which browser it opens (note that you cannot use `$authorize()` with a thick client task when running in the headless server). It should be noted that when running in the thick client, `$authorize()` always uses a web browser rather than an embedded obrowser control due to best practice considerations documented in RFC 8252: <https://www.rfc-editor.org/rfc/rfc8252.txt>

When executed within the context of a **remote task**, `$authorize()` will only work if the remote task is a JavaScript Client remote task. In this case, it uses the `$showurl()` mechanism of the JavaScript Client to open a browser window or tab. Note that in this case, you cannot execute both `$authorize()` and `$showurl()` in response to the same JavaScript client event.

When using the authorization flows that redirect the browser to a URI, `$authorize()` determines the redirect URI as follows.

For the **thick client**, it uses a loopback URI, to 127.0.0.1. Note that if the version of Omnis is not a server version, Omnis will still open a server port with limited support for OAuth2 only, to allow the Authorization Code to be received via the redirect URI.

For the **JavaScript client**, `$authorize()` uses the RESTful URI determined from the Omnis server configuration. Note that this means that if you are using a Web Server to handle requests for your Omnis server, you need to set up the Omnis Web Server plugin for both the JavaScript client and RESTful requests.

`$authorize()` takes a single parameter, `iAuthFlow`, which can have one of the following constant values (`kOW3OAUTH2authFlowCodeWithPKCE` is the default):

Constant	Description
<code>kOW3OAUTH2authFlowCode</code>	The normal OAuth2 authorization flow, where the authorization code will be received by redirecting the browser to a URI served by Omnis.

kOW3OAUTH2authFlow CodeWithPKCE	Identical to kOW3OAUTH2authFlowCode, except that the worker uses PKCE to further secure its requests for an authorization code; the default iAuthFlow (see https://tools.ietf.org/html/rfc7636).
kOW3OAUTH2authFlow ManualCode	Like kOW3OAUTH2authFlowCode, except that the redirect URI is urn:ietf:wg:oauth:2.0:oob. This means that instead of the authorization code arriving at Omnis via the redirect URI, the user must copy the authorization code to the clipboard from the browser window, and paste it into Omnis or the JavaScript client; after pasting, the Omnis application must call the method \$setauthcode() (described below).
kOW3OAUTH2authFlow ManualCodeWithPKCE	Like kOW3OAUTH2authFlowManualCode, but also uses PKCE.

Note that you would normally use PKCE unless the service does not support it.

Manual code support, via the clipboard, is provided in case you do not want to open up a port for the redirect URI when running in the thick client; however, note that not all services support the redirect URI urn:ietf:wg:oauth:2.0:oob.

When \$authorize() completes (which if successful means that it has opened the browser, received the Authorization Code, and exchanged it for an Access Token etc) it generates a call to the callback method \$completed().

\$setauthcode

`$setauthcode(cAuthCode)`

Returns Boolean true for success.

Only applicable to kOW3OAUTH2authFlowManualCode and kOW3OAUTH2authFlowManualCodeWithPKCE, when the \$authorize() thread is waiting for the Authorization Code. Called from the application to supply the pasted Authorization Code using the cAuthCode parameter.

\$save

`$save(&xOAUTH2[,xKey])`

Saves the properties (\$clientid, \$clientsecret, \$authorizeurl, \$tokenurl, \$scope, \$accesstoken, \$refreshtoken and \$accesstokenexpiry) to the encrypted binary buffer xOAUTH2.

xKey is a 256 bit AES encryption key. If you omit xKey, OW3 uses a hard-coded default key.

Returns Boolean true for success.

\$save provides a convenient way to save all of the OAUTH2 parameters required for communicating with a service. In particular, it lets you safely store the Refresh Token, so you can minimise the number of occasions on which a user needs to authorize access using \$authorize().

You can further protect your client secret, by including the encrypted buffer generated by \$save in your release tree.

\$load

`$load(xOAUTH2[,xKey])`

Loads the properties (\$clientid, \$clientsecret, \$authorizeurl, \$tokenurl, \$scope, \$accesstoken, \$refreshtoken and \$accesstokenexpiry) from the encrypted binary buffer xOAUTH2 previously generated using \$save().

xKey is a 256 bit AES encryption key. If you omit xKey, OW3 uses a hard-coded default key. You must use the same key as that used when calling \$save().

Returns Boolean true for success.

OAUTH2 Callback Methods

\$tokensrefreshed

The OAUTH2Worker has one non-standard callback method, \$tokensrefreshed. The OAUTH2Worker generates a call to this method after it has successfully refreshed the tokens while it is being used in conjunction with the HTTP/IMAP/POP3/SMTP worker. \$tokensrefreshed() is called with no parameters; at this point, the worker has been updated with the new Access Token, Access Token Expiry and Refresh Token. A typical implementation of \$tokensrefreshed() would use \$save() to save the current tokens etc and then write the encrypted buffer to disk. It should be noted that calling the server to refresh tokens can result in a different updated Refresh Token - this needs to be used to refresh tokens the next time a refresh is required.

HTTP and General Methods

The OAUTH2Worker supports the normal methods \$cancel(), \$getsecureoptions() and \$setsecureoptions(). The latter two relate to how secure connections to the Token URL are established.

HTTP Callback Methods

The OAUTH2Worker generates calls to the standard callback methods \$cancelled() and \$completed(). These correspond to a call to \$authorize() to start the authorization code flow. The completion row passed as a parameter to \$completed() has columns as follows:

Column	Description
errorCode	An integer error code indicating if the request was successful. Zero means success. If successful, \$accesstoken, \$accesstokenexpiry and \$refreshtoken have been updated using the content received from the server; if no Access Token Expiry was received, \$accesstokenexpiry is #NULL; if no Refresh Token was received, \$refreshtoken is empty.
errorInfo	A text string providing information about the error if any.
scope	The scope returned from the server when requesting the Access Token, if different to the requested scope.
log	If you used \$protocollog to generate a log, this column contains the log data, either as character data, or UTF-8 HTML. Otherwise, the log column is empty.

HTTP, IMAP, POP3, and SMTP Workers

Once you have used \$authorize() to obtain an Access Token, you need to make the Access Token available to the worker with which OAUTH2 authorization is required. You do this by assigning a new \$oauth2 property of the HTTP, IMAP, POP3, or SMTP worker:

❑ \$oauth2

Property that is an object reference to an OAUTH2Worker object containing the authorization information required to make requests to the server. Clear this property by assigning #NULL to it. \$authorize() cannot run while the OAUTH2Worker is assigned to \$oauth2

The supported workers use \$oauth2 to obtain the Access Token for the request. To do this, it uses the following logic:

- ❑ If there is no Refresh Token (\$refreshtoken is empty), it uses \$accesstoken.
- ❑ If the \$accesstokenexpiry is #NULL (there is no expiry date and time), it uses \$accesstoken.
- ❑ If the expiry date time is more than 5 seconds away, it uses \$accesstoken

- ❑ Finally, it uses `$refreshToken` to refresh the token(s). If successful, it generates a call to `$tokensRefreshed()` in the `OAUTH2Worker` and it uses the new `$accessToken`. You should note that there is a chance the request will fail when it is made near to the 5 second window before the Access Token expires. You should be prepared to handle this type of error in `$completed`, possibly retrying the request.

HTTP

After assigning `$oauth2`, the parameters `iAuthType`, `cUserName`, and `cPassword` passed to `$init()` are ignored in favour of using the Access Token stored in `$oauth2`.

IMAP, POP3, SMTP

After assigning `$oauth2`, the `cPassword` parameter passed to `$init()` is ignored in favour of using the Access Token stored in `$oauth2`. Note that `cUserName` is still required.

OW3 Worker Request Completion

There is a new property, `$alwaysfinish`, in the OW3 Worker Objects external package, to allow asynchronous requests to continue to completion after the instance containing the OW3 object destructs; the property applies to the HTTP, IMAP, SMTP, POP3 and FTP workers.

When the instance containing an OW3 worker closes, and the OW3 worker is executing via a call to `$start()`, the worker thread continues executing until completion in the background: in this case, no notifications will be generated, as there is not a suitable instance to receive them.

Note that even if `$alwaysfinish` is true, if you shut down Omnis before the request has completed, OW3 will cancel the request so that shutdown works correctly.

Multipart Content

There is a new static method, `$splitmultipart()`, in the OW3 Worker Objects external package to allow you to split multipart content of a rest call, plus the MIME list returned by the OW3 methods that contain body part headers.

- ❑ **OW3.\$splitmultipart**(`cContentType`, `xContent`, `&IMIMEList` [`,iDefCharSet=kUniTypeUTF8`, `&cErrorText`])
 splits MIME-encoded multi-part `xContent` into `IMIMEList`. `cContentType` must include a boundary parameter. Returns true if successful
`ContentType`:The content type header (must contain a boundary parameter)
`Content`:The binary content to split
`MIMEList`:Receives the MIME list created by splitting the MIME content. See the documentation for the `MailSplit` command to see how a MIME list is structured;however note that the charset in the OW3 MIME list is a `kUniType...` constant
`DefCharSet`:The default character set used to convert character data when there is no charset specified for a MIME text body part. A `kUniType...` constant (not `Character/Auto/Binary`)
`cErrorText`:If supplied, receives text describing the error that caused `$splitmultipart` to return false

The MIME list (for this call and for the other OW3 calls that generate a MIME list) now contains an additional column named `bodyparheaders`. This is a row containing a column for each non-empty header present for the body part. In addition, it has a column named "name" which contains the content-disposition header name parameter. All header names are normalized in the same way as those passed to RESTful services, that is, lower-case with any - characters removed.

IMAP Worker

The OW3 IMAP worker now allows you to fetch only unread messages. An IMAP search query parameter has been added to the list messages action, to control the list of messages in the mailbox that is returned.

The `kOW3imapActionListMessages` action now selects mailbox `cMailboxName` and lists messages in it. `vParam1` (optional) is a single column list of additional header names to retrieve. `vParam2` (optional) is an IMAP search query selecting messages to list, e.g. UNSEEN to fetch the unread messages.

FTP Worker

New actions have been added to the OW3 FTP worker to allow you to send multiple files/folders via an FTP client, as follows:

❑ **kOW3ftpActionPutFileMulti**

Upload multiple files to the FTP server. `vParam` is a 2 column list (`col1`: full local pathname, `col2`: full server pathname). `cServerPath` must be empty

❑ **kOW3ftpActionGetFileMulti**

Download multiple files from the FTP server. `vParam` is a 2 column list (`col1`: full local pathname, `col2`: full server pathname). `cServerPath` must be empty

When uploading or downloading multiple files, the row passed to `$progress` has an extra column (`requestNumber`) which corresponds to the line number in the `vParam` list currently being transferred.

The `$completed` method is called with a successful status if all transfers are completed successfully. If at least one failed, the error code is 10312 (at least one transfer during a `kOW3ftpActionGetFileMulti` or `kOW3ftpActionPutFileMulti` action failed). In addition, the `resultList` column contains a list with a line for each transfer, containing error code, error info and FTP status code.

HTTP Worker

The minimum number of parameters for `$init()` method in the OW3 HTTP worker is now 1, rather than 4 in previous versions, so the URI is the only required parameter. The definition for the method is now:

```
$init(cURI [,iMethod=kOW3httpMethodGet, IHeaders=#NULL, vContent=",
iAuthType=kOW3httpAuthTypeNone, cUserName=",cPassword="])
```

Mail Headers

The character limit of 76 for RFC2047 encoded words for mail headers has been removed in the email OW3 workers (and the `MailSplit` command).

Web Services

OpenAPI

Omnis now generates an OpenAPI 3.0.0 definition for a RESTful service as well as Swagger 2.0. OpenAPI is a more up to date version of the RESTful API description format, and Studio 10.2 now generates OpenAPI 3.0.0 definitions, as well as Swagger 2.0 definitions.

When you select a RESTful service beneath the Web Service Server node in the browser, there are now two pairs of links:

- OpenAPI Definition, Save OpenAPI to File
- Swagger Definition, Save Swagger To File

The OpenAPI definition can be retrieved using a similar URL to that used to retrieve a Swagger definition by replacing 'swagger' in the URL with 'openapi'.

There is a new folder in `clientserver/server/restful`, named `openapitemplates`. The files in here have the same use as those in the `swaggertemplates` folder, except that they apply to OpenAPI definitions.

In addition, `cors.json` has new OpenAPI members that have a similar purpose to the Swagger members.

In previous versions, you could provide a format by prefixing a description of a schema field or HTTP method parameter with "`<swagger-...>`". In Studio 10.2, you can now provide a format using the prefix of either "`<format-...>`" or "`<swagger-...>`".

Media types

HTTP responses for a RESTful method can now be defined to return media types other than `application/json` via a schema.

Note that if you do this, the Swagger 2 definition is incomplete, since Swagger 2 does not allow mixed response content types. However, the new OpenAPI 3 definition for the RESTful service does handle this correctly.

\$construct parameter row

The full URL used to request a RESTful method is now passed via RESTful remote task \$construct parameter row.

There is a new column in this row: `fullurl`, which contains the full URL, starting with the path to the script, e.g. `/omnisrest/ws/5988/api/...`

The host name used can be obtained from the host header. There is no way to determine if the request was made using `http` or `https`.

RESTful Remote Task Superclass

A RESTful remote task can now have a superclass in another library, provided that the superclass in the other library does not contain URIs. This allows you to use framework libraries.

Object Oriented Programming

\$cando and Error Handling

In previous versions there were issues when using \$cando with object variables with a missing object class, and also with object variables and the debugger.

The default behavior (since Studio 10.1), when Omnis attempts to construct an object variable when its class does not exist, is to report a debugger error when code attempts to use the object, e.g. via \$cando. Therefore, you can now override this behavior, for example, \$cando will return kFalse for notation like `iObject.$message.$cando`.

To override this behavior, you can use the new \$nofatal property of object variables:

```
Calculate iObject.$nofatal as kTrue
```

❑ \$nofatal

If true, and the object instance could not be constructed because the object class does not exist, treat this error as a warning when trying to use this object (meaning Omnis will not enter the debugger or abort execution)

When using \$cando after setting \$nofatal to kTrue, the \$cando will return false.

\$inherited and \$default

The notation \$inherited and \$default have been added to the help displayed in the Code Assistant (they were hidden in previous versions).

\$inherited is now present at the top level, and in \$cinst for all types of instance, while \$default is available at the top level, and in \$cinst but for table instances only.

In addition, you can now use \$default.\$attrib() as well as \$cinst.\$default.\$attrib().

External Objects

The \$isa() function now works for external objects.

JSON Components

JSON Control definition

There is a new member "uselegacycolors" for a JSON control definition; it is automatically set to True when loading existing JSON controls so the existing colors are used. The flag defaults to False for all new controls which means they can use theme colors.

Commands

Queue Commands

The *Queue click*, *Queue double click*, *Queue scroll* and *Queue set current field* commands (and their obsolete command equivalents) now return an error if the field cannot be found. You can turn off this error by setting the new option "reportQueueCommandFieldNotFoundErrors" to false, located in the "defaults" section of config.json.

Functions

FileOps.\$selectfilesinsystemviewer

There is a new static method in the FileOps external \$selectfilesinsystemviewer to allow you to open a file or files (such as a library) in the system File Explorer in Windows or the Finder on macOS.

```
FileOps.$selectfilesinsystemviewer(cFileOrFolder[,lFileList])
```

Opens the system file viewer (Explorer or Finder) and selects the specified file or files.

- ❑ **cFileOrFolder**
Either the pathname of a single file to select (when \$selectfilesinsystemviewer is called with a single parameter) or the pathname of the folder containing the files to select (when \$selectfilesinsystemviewer is called with 2 parameters)
- ❑ **lFileList**
A single column list of file names, specifying the files to be selected in the specified folder

FileOps.\$writecharacter()

There is a new parameter in the FileOps.\$writecharacter() function that can be used to control whether or not a BOM is added to the data when writing to the start of the file.

- ❑ **\$writecharacter(iEnc,cData[,bAppend=kFalse,bBOM=kTrue])**
Writes cData to file, encoded using encoding iEnc (kUniType... but not Auto/Bin/Char). Returns kTrue for success.
bAppend kFalse means replace entire file contents, if kTrue data is appended to the existing data.
bBOM controls if a BOM is added or not. If true, and the data is to be written at the start of the file, a Unicode Byte Order Marker (BOM) is added at the start of the file.

FileOps.\$deletefile

The FileOps.\$deletefile function has two new parameters, deleteContents and recursive (both default to kFalse, if omitted), to allow you to delete the contents of a folder.

- ❑ **FileOps.\$deletefile(cPath, kFalse, kFalse)**
(same as if deleteContents and recursive are omitted) if cPath is a folder, the folder will be deleted only if empty, that if no files or directories are present in the folder. On macOS, the function checks if the only file is .DS_Store and will delete the folder in that case. If there are hidden files in the folder, the folder and the hidden files will not be deleted.
- ❑ **FileOps.\$deletefile(cPath, [kTrue, kFalse])**
if cPath is a folder, *all files* inside the folder if deleteContents is true, but any subfolders *will not* be deleted. Furthermore, if the folder is empty, the folder will not be deleted.
- ❑ **FileOps.\$deletefile(cPath, kFalse, kTrue)**
if cPath is a folder, it will be deleted only if empty. If there are any contents, those will not be deleted as deleteContents is kFalse, even if recursive is kTrue.
- ❑ **FileOps.\$deletefile(cPath, kTrue, kTrue)**
if cPath is a folder, it will be wiped off the disk with its contents, if any are present.
- ❑ **FileOps.\$deletefile(cFilePath)**
if cPath is a file, the file will be removed regardless of the deleteContents or recursive values - the deleteContents and recursive parameters do not impact the behavior of deleting files, only the behavior of deleting folders.

FileOps.\$createdir()

The FileOps.\$createdir() functions has a new optional boolean parameter bCreateParentDirs. If bCreateParentDirs is kTrue, FileOps will create the directory in cPath and also any parent directories in the path that do not exist.

If bCreateParentDirs is kTrue and the directory in cPath already exists, FileOps will return 0 (success) rather than 101215 (A file with the specified name already exists). If bCreateParentDirs is kFalse (default), or is omitted, then the usual 101215 is returned if directory in cPath already exists.

rxpos()

With the introduction of PCRE2 the rxpos() function has been enhanced.

A new optional final parameter called captureRow has been added to the rxpos() function (applies when PCRE2 is used, which is now the default for regular expressions). You can pass this a row variable that returns the captured groups resulting from the regular expression match operation. This is a standard feature of regular expressions – groups correspond to the parts of the regular expression contained in parentheses (provided that the open parenthesis is not followed by ?: indicating a non-capturing group). Groups can also optionally be named, and they are numbered 1 to n, with various rules regarding duplication when using the | operator.

For example:

- ❑ When rxpos() locates nothing, it sets captureRow to empty.
- ❑ When rxpos() locates something, and the captureRow parameter is supplied, it adds a column to the captureRow for each captured group, where the column name is G<n> for group number n where the group is not named, or the group name.

The following examples will illustrate this:

Calculate cString as "2017-01-02"

```
Do rxpos ("^(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})$", cString, 0, 0, cLen, cRow) Returns cOffset
```

After executing the above line, cRow is a row with three columns named year, month, and day, with the values 2017, 01 and 02, respectively.

Calculate cString as "hey_test_ho"

```
Do rxpos ("(hey|ho)_test_(ho|hey)", cString, 0, 0, cLen, cRow) Returns cOffset
```

After executing the above line, cRow is a row with two columns named G1 and G2, with the values hey and ho.

You can mix named and unnamed capture groups.

binfrombase64()

A new parameter, bStripWhitespace, has been added to the binfrombase64() function, to strip whitespace from the input data (defaults to kFalse). The syntax for the function is now:

- ❑ **binfrombase64**(vData[,bURLEncoding=kFalse,bExpectPadding=kTrue,bStripWhite space=kFalse])

There was a problem using the OXML base64 method (which is essentially deprecated) so you should use this function instead, which is also faster. In addition, the old implementation in OXML ignores whitespace in the input data, hence you are advised to use binfrombase64() with the new parameter.

printf()

There is a new function printf() to allow you to output a string from a script:

```
printf(string[, newline=kTrue])
```

Writes the string to standard output followed by a newline character if required (the function is ignored on Windows. Executes on macOS and Linux only).

mod()

mod() now checks for integer arguments, and if so uses integer rather than floating point operations to determine its result. The result is still returned as a number rather than an integer.

sys(202)

The sys(202) function returns the command line parameters passed to Omnis.

In order to pass arguments to omnis.app using open, you can use --args, for example, to start Omnis and open a library at startup you can pass the library name as the first argument using:

```
open -W -a omnis.app --args /library.lbs
```

If the first argument passed via the command line is not a library, you can prepend it with a - (hyphen) character. So when using sys(202), you will get all arguments passed to Omnis, including the first argument, whether it starts with a hyphen or not.

sys(241)

A new 6th column named 'item' has been added to list returned by sys(241), the find and replace log list. Where possible, this column contains an item reference either directly to the found data, such as a method line, or to the item containing the found data, such as a property.

sys(250)

The **sys(250)** function has been added which returns a list of files which were used to open Omnis, e.g. double-clicked from the Finder or passed on the command line. This is empty if Omnis was opened directly by double-clicking.

OJSON

OJSON.\$arrayarraytolist

OJSON.\$arrayarraytolist now allows integer and numeric values in the same column (resulting in a numeric column).

JavaScript API

jOmnis object methods

All supported browsers now support RGBA colors so any related methods are no longer relevant and have been removed. The following methods have been removed from the jOmnis object:

- browserSupportsRGBA
- startTranslateAnimation
- inIE - removed as it returns false for all supported browsers
- ieVersion - removed as the JS client only supports IE 11 upwards
- setOpacity - pSetFilter parameter was removed in a previous version

Import/Export

Delimited Import

Delimited import now allows newline characters to be embedded within delimiters.

Omnis VCS

VCS Revisions

All class types now have the `$vcsrevision` property, which allows the Omnis VCS to determine whether or not classes in a local library are up to date with the latest revision in the VCS repository.

VCS revision property

The `$vcsrevision` property is the revision number of the class, and is used for classes stored in the Omnis VCS. It is set to zero if the library was not built by the VCS (i.e. it was created in the IDE), or if the library was built prior to Omnis Studio 10.2.

The property is read-only in the Property Manager, as it is intended for use by the Omnis VCS library, or any custom class editing tools you may have created. It can only be assigned by executing code.

Server Connections

When the VCS loses its network connection, the list of classes and all hyperlink options are hidden and replaced with a **Refresh** option (previous versions may have issued numerous messages when a connection was lost). Pressing Refresh will poll the database to see if the connection has been restored, but if the connection is still down an error message will be shown. Polling the database may still result in a wait of several seconds before the connection is restored or any error message is shown.

Deployment

Server port

The `$serverport` property has been added to the `$modes` group (`$modes.$serverport`) and returns the port on which the Omnis Server is currently listening.

Printing JPGs on Headless Server

In order to print JPEGs from an application running on the Headless Omnis Server (on Linux), the ImageMagick package has to be installed.

Headless Server Logging

The Headless Server now logs a message when an external or external component cannot be loaded. This is a message of type `headlesserror`, and includes the system error text reporting the missing dependency that caused the component not to open.

Windows Startup Options

The Omnis startup options previously set in the `Omnis.ini` file have been moved into the Omnis configuration file (`config.json`) under the Windows group and the `omnis.ini` no longer works. Therefore, you can now specify the following under the windows group:

```
"HideStudiorgMessage": false,  
"NoAdmin": false,  
"UpdateFilesAssociations": true
```

If `HideStudiorgMessage=true` the message dialog about running Studiorg when Omnis starts up will not be displayed. If false or omitted, the message is shown.

If `NoAdmin=true`, 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=false` Omnis will run as the Admin user (the default behavior, as in previous versions).

If `UpdateFileAssociation=false`, Omnis will not attempt to set file associations.

Omnis Datafile Migration

DML Emulator

The DML emulator was released with Studio 10.0 but has been substantially re-written for Studio 10.2 to improve performance. We would like to thank early adopters Nick Renders, Thad Bogert, and Martin Luce for helping us to develop and test the new emulator.

The DML emulator no longer relies on library code for most of its functionality, although it is still used for the initial conversion of the data file to either a SQLite data file or a PostgreSQL database.

In order to prevent conflicting emulation modes, there is a new `$root` preference; `$mapdmltodam` which is set to either "PGSQLDAM" or "SQLITEDAM" to set the emulation mode. The existing library preference has now become `$dmlemulation`, a Boolean property which enables that library for DML emulation.

Please note that switching emulation modes will shut down the emulator, closing any connections before re-initializing in the new mode.

What is DML Emulation?

In summary, if you have an older Omnis application that uses Omnis data files and the Data Manipulation Language (DML) commands such as *Set main file*, *Find*, *Next & Previous*, *Build list from file...*, you can now perform a one-time conversion of your Omnis data file(s) to a PostgreSQL database or to a SQLite data file. By setting the Studio root preference and the library preference mentioned above, your library continues to execute as it did before, without any code modifications, but future-proofed against any potential pitfalls with legacy Omnis data files.

Porting your data to a proprietary database also makes it more accessible to third-party applications.

For further information on the DML emulation technology, please refer to the 'Omnis Datafile Migration' chapter in the Omnis Programming manual on the Omnis website.

External Components

There has been a number of changes in the External Components interface, so please refer to the [online manual](#) for full details. The following functions or items have been changed or added:

- `WNDstartDrawEx()`
- `WNDendDrawEx()`
- `WNDstartDraw()`
- `EXTfile::deleect()`

What's New in Omnis Studio 10.1

Omnis Studio 10.1 contains a number of enhancements in the Code Editor (introduced in Studio 10.0), plus some updates for JavaScript Remote forms and various JS controls. The following features and enhancements have been added to Omnis Studio 10.1:

- ❑ **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

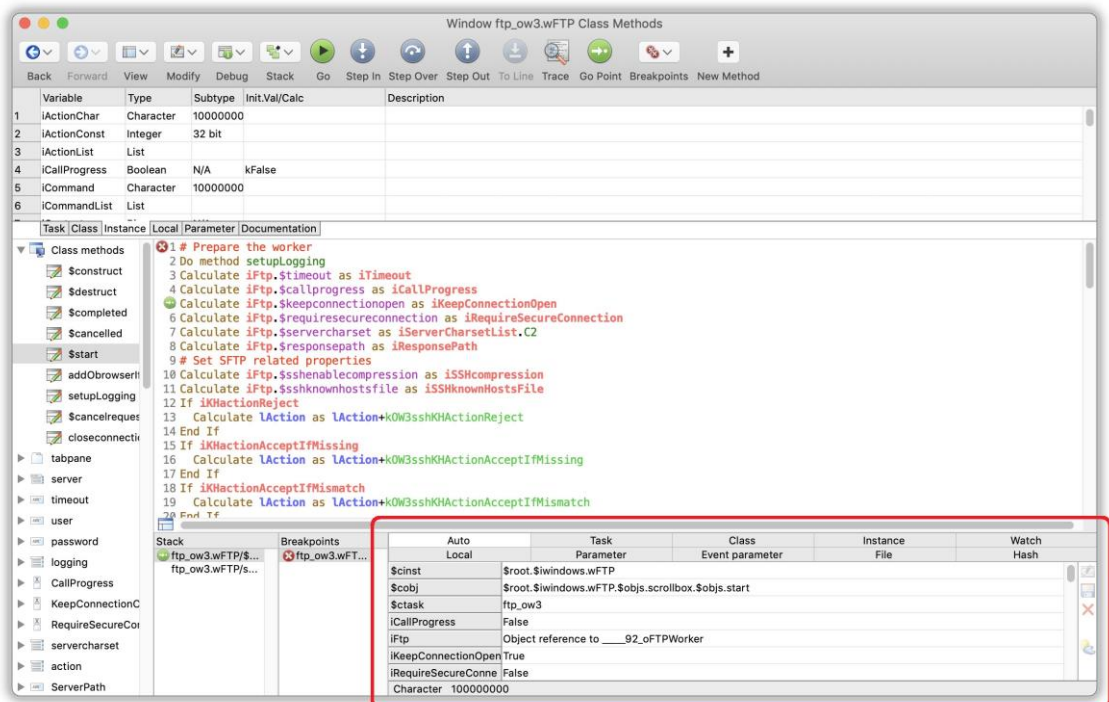
The Code Editor (Method Editor) introduced in Studio 10.0 has had a number of enhancements, including several in response to customer feedback.

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:

Auto		Task		Class		Instance	
Local	Parameter	Event parameter	File	Hash			


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:


	iObjectRef	Class	Instance
	Object reference to ___132_oTimerWorker	Properties	Task

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:

	iList	Next
	Lines 1 to 64 of 10000	Previous

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.

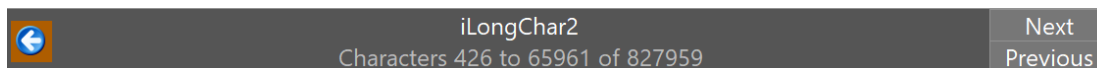
You can modify the current line and selection of the list using the buttons on the variable panel. These prompt for the new current line, or changes you want to make to the selection.

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 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 condition	setBreakpointCondition
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

Windows shortcut	macOS shortcut	Description	Keys.json item
Ctrl+/ Ctrl+[Cmnd+/ 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
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

Windows shortcut	macOS shortcut	Description	Keys.json item
Ctrl+Shift+M	Cmd+Shift+M	Superclass methods	superclassMethods
Ctrl+Shift+N	Cmd+Shift+N	Show method tree	showMethodTree
Ctrl+Shift+O	Cmd+Shift+O	Toggle one-time breakpoint	toggleOneTimeBreakpoint
Ctrl+Shift+R	Cmd+Shift+R	Previous error	previousError
Ctrl+Shift+S	Cmd+Shift+S	Step	step
Ctrl+Shift+T	Cmd+Shift+T	Step out	stepOut
Ctrl+Shift+U	Cmd+Shift+U	Upper case selection	upperCaseSelection
Ctrl+Shift+V	Cmd+Shift+V	Step over	stepOver
F1	F1	Opens the Omnis Help using the syntax item <i>under</i> 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 <i>under</i> 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 $\geq 0x250$ in the Code Editor has been improved.

The Code Editor now selects a smaller font size, if necessary, for all Unicode characters $\geq 0x250$ 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` / `Cmd+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	Show fields OBSOLETE COMMAND
Clear window control method OBSOLETE COMMAND	SNA do not perform default action OBSOLETE COMMAND
Disable fields OBSOLETE COMMAND	SNA perform a Cancel OBSOLETE COMMAND
Enable fields OBSOLETE COMMAND	SNA perform a shift-tab OBSOLETE COMMAND
Hide fields OBSOLETE COMMAND	SNA perform a tab OBSOLETE COMMAND
Queue click OBSOLETE COMMAND	SNA perform an OK OBSOLETE COMMAND
Queue double-click OBSOLETE COMMAND	SNA perform command OBSOLETE COMMAND
Queue scroll OBSOLETE COMMAND	SNA perform default action OBSOLETE COMMAND
Queue set current field OBSOLETE COMMAND	SNA remain on current field OBSOLETE COMMAND
Redraw named fields OBSOLETE COMMAND	SNA set current field OBSOLETE COMMAND
Redraw numbered fields OBSOLETE COMMAND	Hide design & commands menus OBSOLETE COMMAND
Redraw windows OBSOLETE COMMAND	Test if command available OBSOLETE COMMAND
Send to a window field OBSOLETE COMMAND	Store window OBSOLETE COMMAND
Set return value OBSOLETE COMMAND	Set palette when drawing OBSOLETE COMMAND
Set task control method OBSOLETE COMMAND	
Set window control method 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 be assigned 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 content 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 \$hilitfocusedcell and \$cellhilitcolor have been added to the JS Data Grid to allow you to highlight the cell that has the focus.

- ❑ **\$hilitfocusedcell**
If true, the focused cell will be outlined in the color specified by \$cellhilitcolor
- ❑ **\$cellhilitcolor**
The color of the focused cell's outline, provided \$hilitfocusedcell 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:

`$toolbarhertextcolor`

The text color of toolbar items when hovered

`$sidemenutextcolor`

The text color of side menu items

`$sidemenuhertextcolor`

The text color of side menu items when hovered

`$overflowtextcolor`

The text color of overflow menu items

`$overflowhertextcolor`

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 column 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, `plnRangeSelection`, 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", "Selecciona un año", "Selecciona
un década", "Selecciona una hora"]
};
```



```
</script>
```

See the Localization chapter in the *Creating 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 thick 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\\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 multibutton 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:

<code>kMsgOK</code>	Display an OK message (the default)
<code>kMsgYesNo</code>	Display a Yes/No message
<code>kMsgNoYes</code>	Display a No/Yes message
<code>kMsgCancelButton</code>	Add a cancel button to the message
<code>kMsgIcon</code>	Display an operating system specific icon with the message
<code>kMsgSoundBell</code>	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 thick 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.

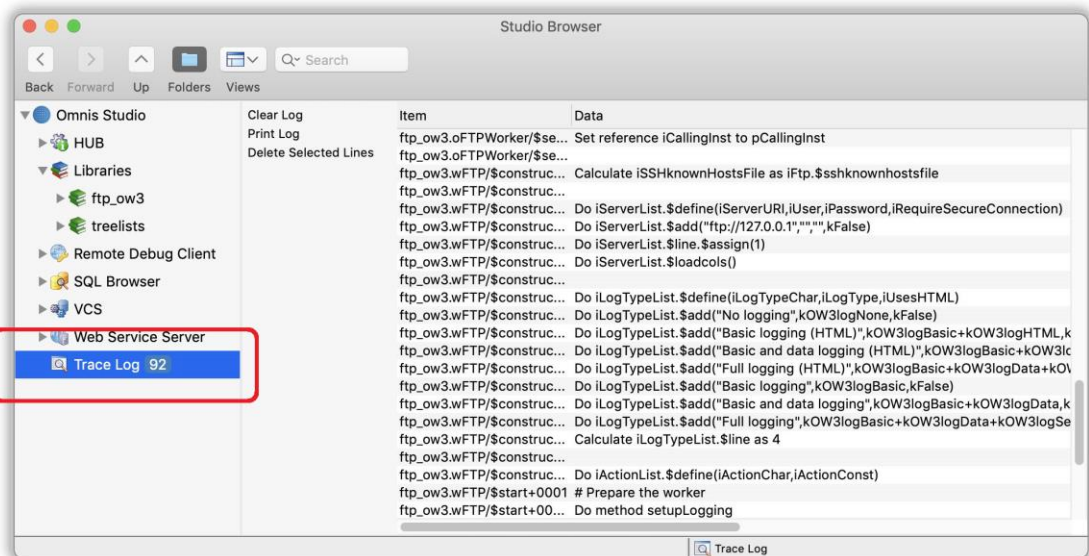
FileOps.\$splitpathname

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 Delimiter

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. `quotedidentifier=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.

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`.

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

\$listorowtojson()

A new `iOptions` parameter has been added to the `$listorowtojson()` 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:

❑ `$listorowtojson()`

`$listorowtojson(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 `$listorowtojson` 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. The maximum key length for a HMAC has been increased from 32 to 64.

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 omnisd.dat 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.

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

- ❑ **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

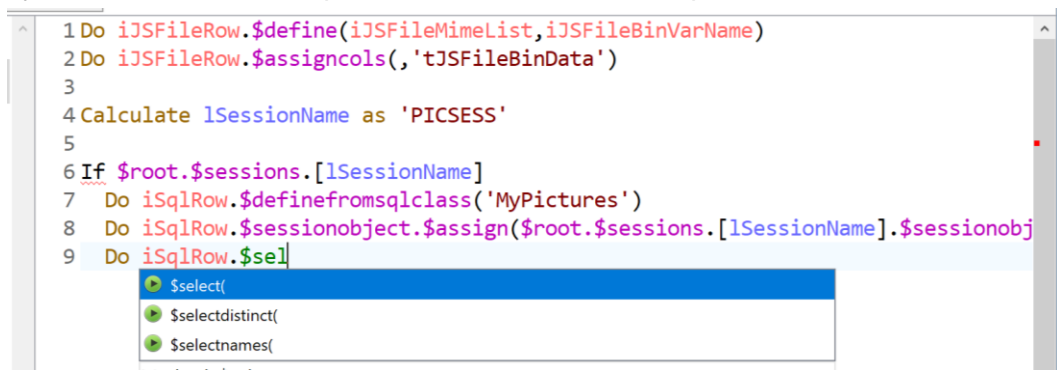
- ❑ **Web and Email Worker Objects**

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.

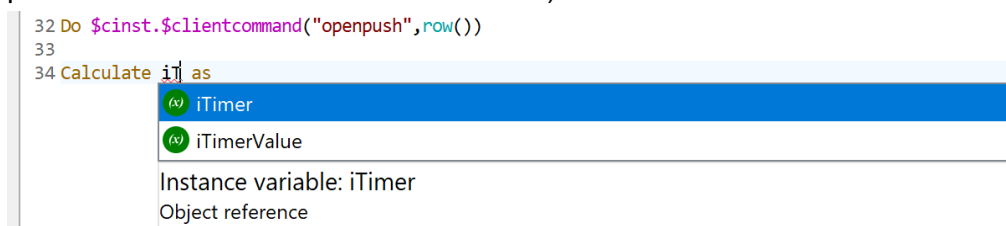
The old Command selection palette, which appeared at the bottom of the Method Editor window, and included the Command list and parameter options, has been removed from the Method Editor. To enter a command, you now tab or click into the first line of a method, then type the first few letters of a command 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 thick 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.

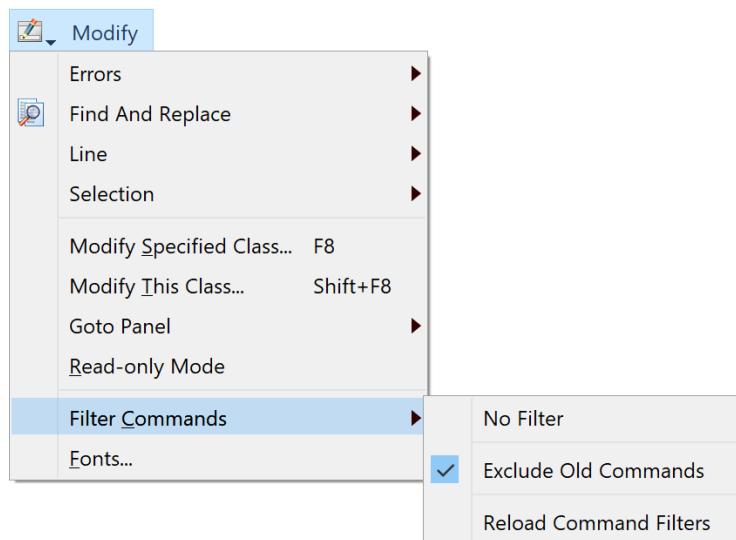
Assuming the cursor is at the end of the selected command name, you can start to enter its parameters, and the Code Assistant should pop up automatically at the insertion point whenever a variable name or parameter is needed.

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".

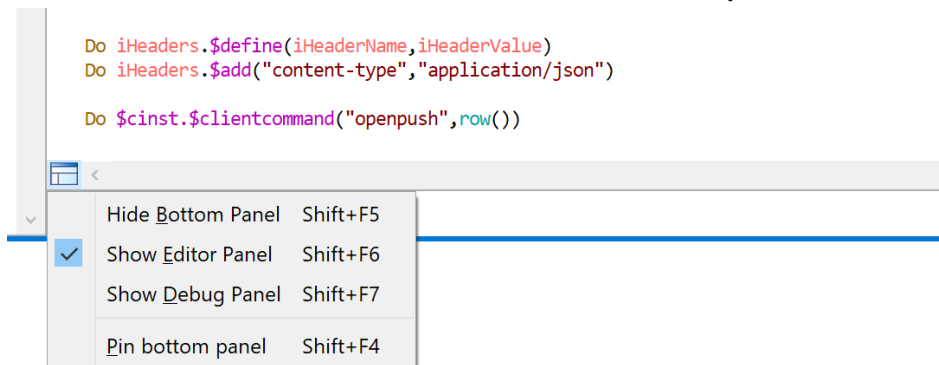
The editor in the background keeps up to date with changes in the foreground editor, e.g. when you add or delete a method, the method list in the other editor updates.

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

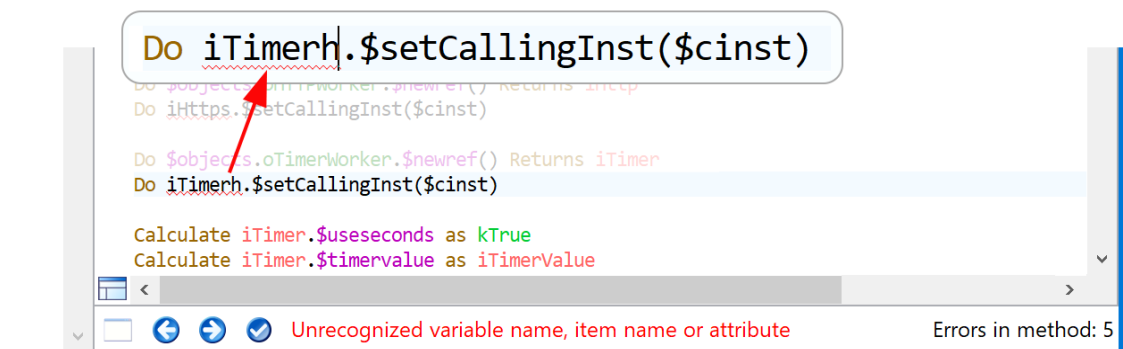
The Method Editor has two panels below the code entry window. The *debug panel* works as it did in previous versions. The *command entry panel* has been replaced with the editor panel.

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 “iDataRow” 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)

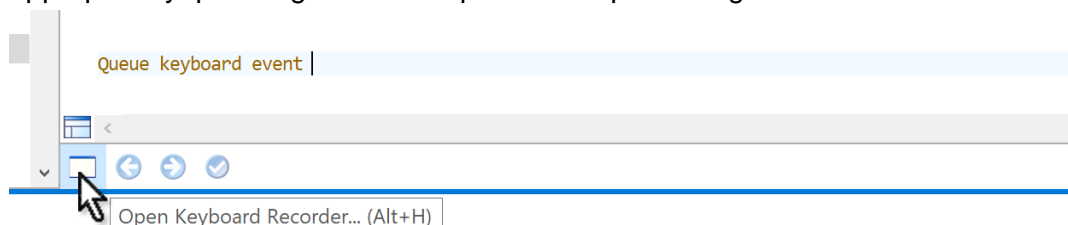
The suffixes allowed in the Create Variable dialog can be configured in the Omnis configuration file (config.json) using a new entry called "createVariableTypeSuffixes" and located in the 'codeAssistant' section in config.json:

```
"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.



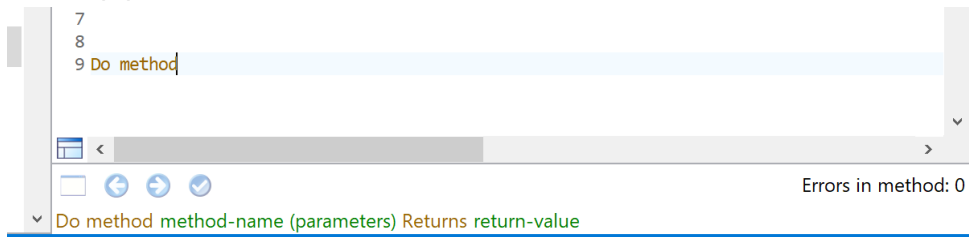
The editor Helper Dialog is enabled in the following cases:

- 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 shown in the Help panel at the bottom of the editor window.



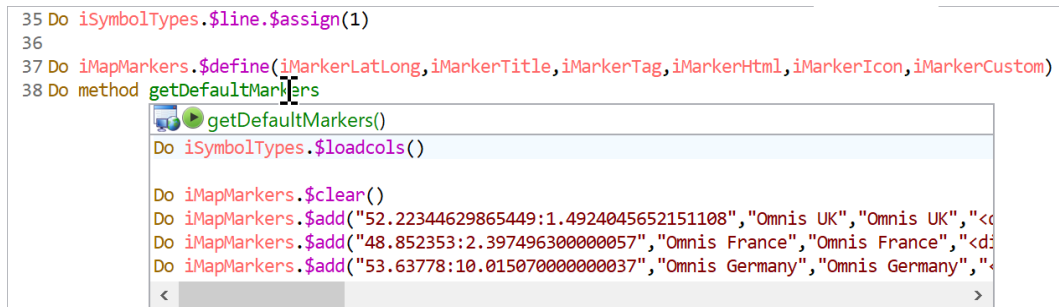
You can hide the command syntax by unchecking the "Show Syntax Strings" option on the View menu.

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.

The following entries in config.json control the size of the Method tooltips:

- "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.

Maximum Number of Methods

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

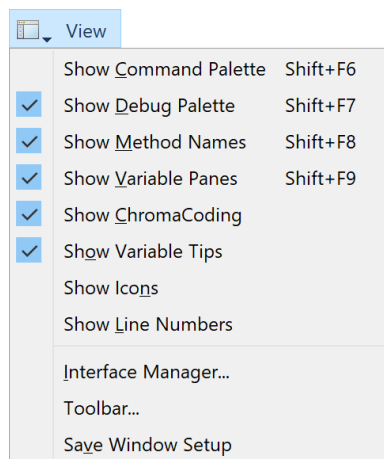
Menus and Keyboard Shortcuts

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

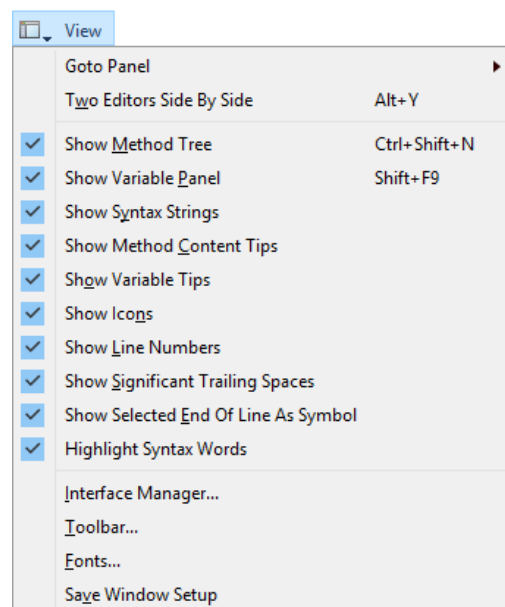
The View menu in the Method Editor has several changes or additions; some of the new options are discussed elsewhere. The **Show Debug Palette** and **Show Chroma Coding** options have been removed; the latter option has been replaced by a more comprehensive set of color options stored in the default 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

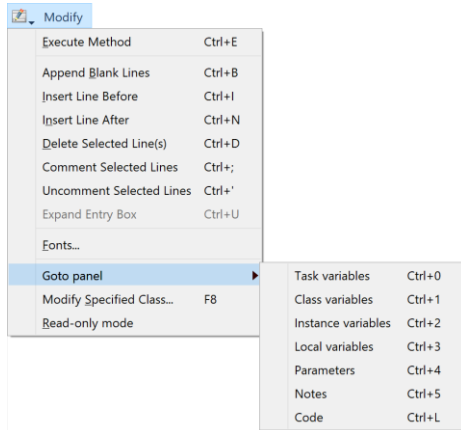


Show Method Tree Ctrl/Cmnd+Shift+N

Goto Panel

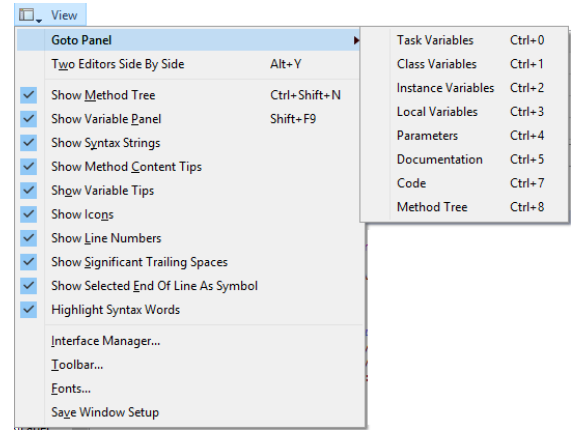
The **Goto Panel** option lets you select a different pane in the Variables list (with keyboard shortcuts Ctrl+0 to 5). It also lets you switch the insertion point to the **Code** text entry area (Ctrl+7) ready to enter some code, or back to the Method Treet list from the code entry area (Ctrl+8).

Studio 8



Goto Code Ctrl+L

Studio 10



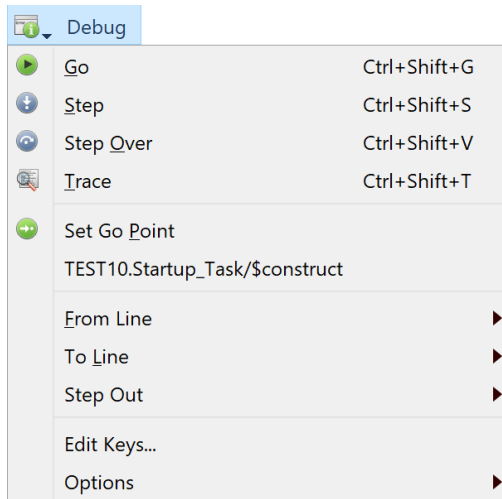
Ctrl+7

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.

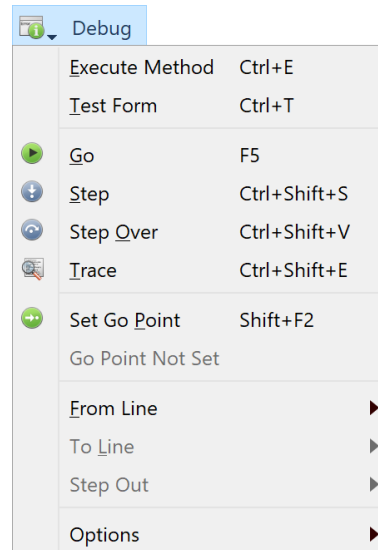
Studio 8



Go Ctrl+Shift+G

Trace Ctrl+Shift+T

Studio 10



F5

Ctrl+Shift+E

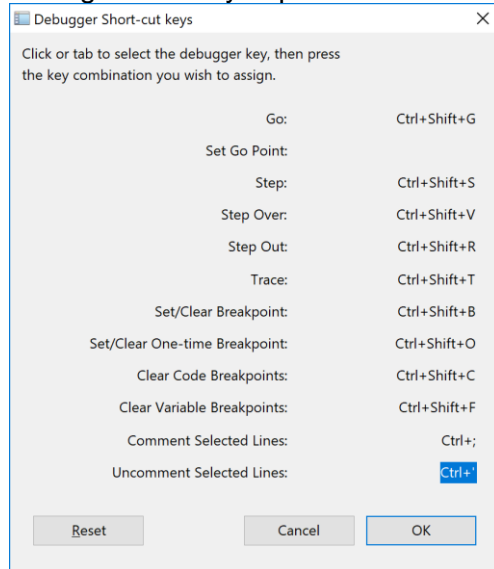
Having entered code using the new text entry method, running the debugger on your your code is exactly the same as in previous versions of Omnis. You can set the Go point, then click Go or Step in to execute your code: as your code executes the

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.

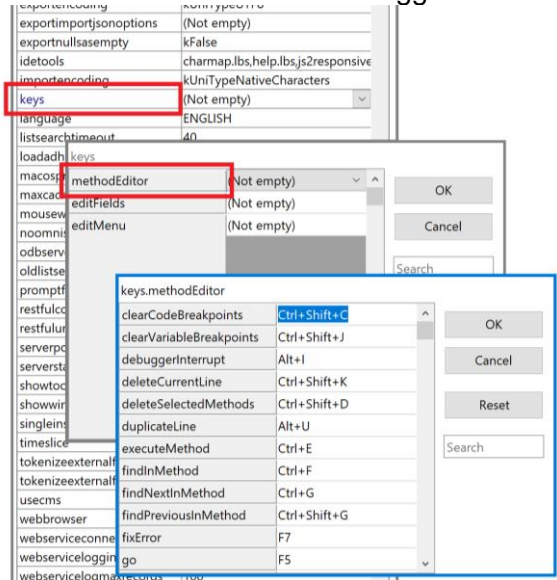
Studio 8

Debug>>Edit Keys option



Studio 10

\$Keys Omnis preference > methodEditorAndRemoteDebugger



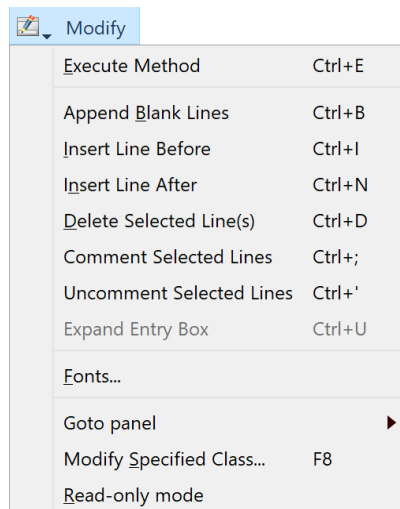
See later in this section for information about changing the keyboard shortcuts.

Modify Menu

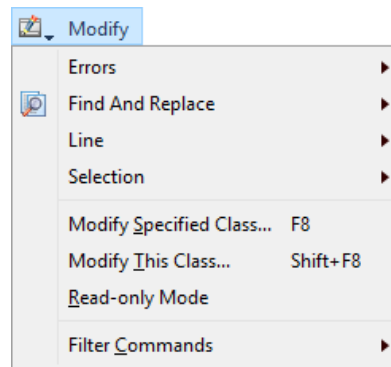
The **Modify** menu contains new submenus for **Errors** and **Find And Replace**. The **Execute Method** option has been moved to the Debug menu, while the Goto panel and Fonts options have been moved to the View menu. The various Line options have been moved to the **Line** submenu.

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

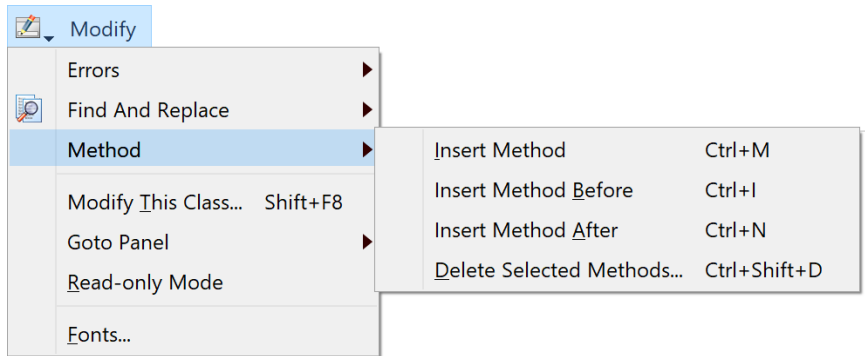


Studio 10



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

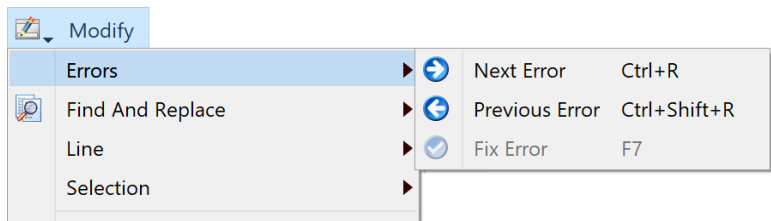
- ❑ If the focus is on the Method Tree (on the left, containing a list of methods for the class), the Modify menu contains a submenu called **Method**, which allows you to **Insert Method** (at the end of the method list, or *Before* or *After* the current method), or **Delete Selected Methods**



- ❑ 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

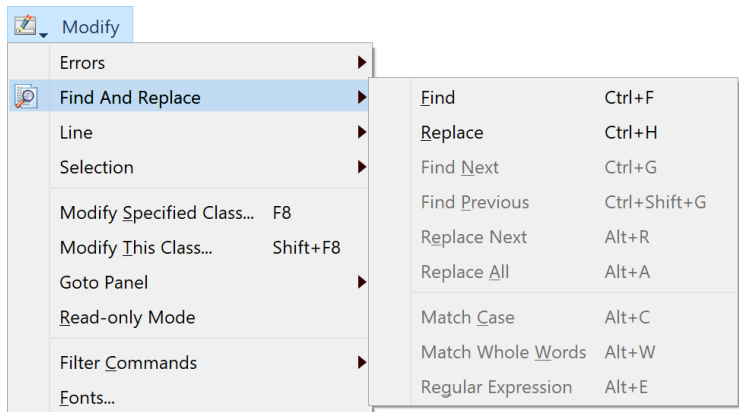
The Modify>>**Errors** submenu is new and contains **Next error**, **Previous error** and **Fix error** commands, that can be used instead of the buttons on the editor panel. These also have keyboard shortcuts.



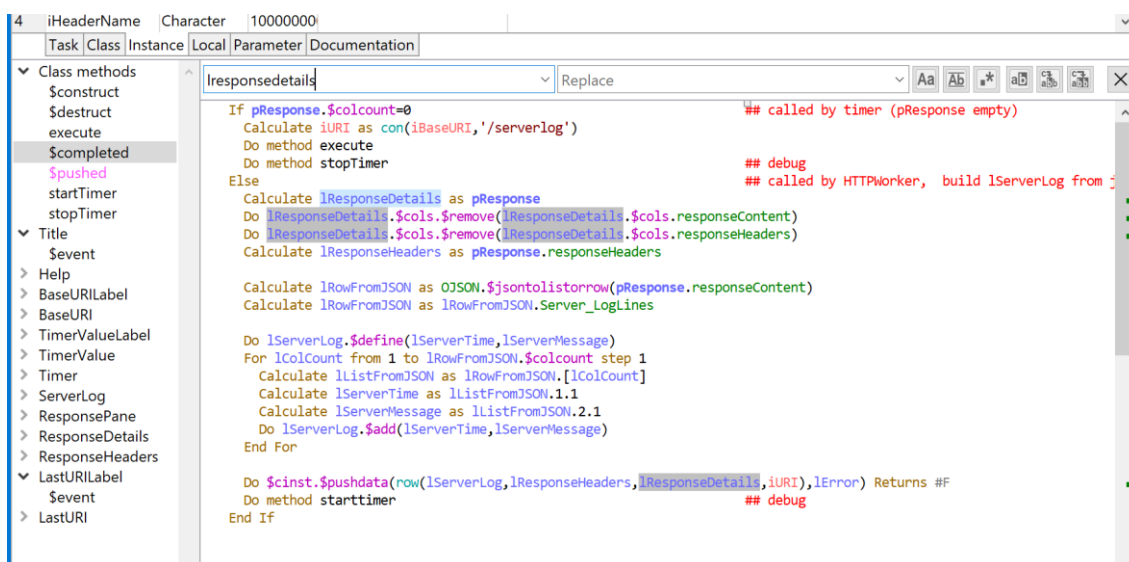
Note that when the focus is on the method tree, this menu is only present when only one method is selected.

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:

	Match case	Alt+C
	Match whole words	Alt+W
	Use regular expressions	Alt+E
	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

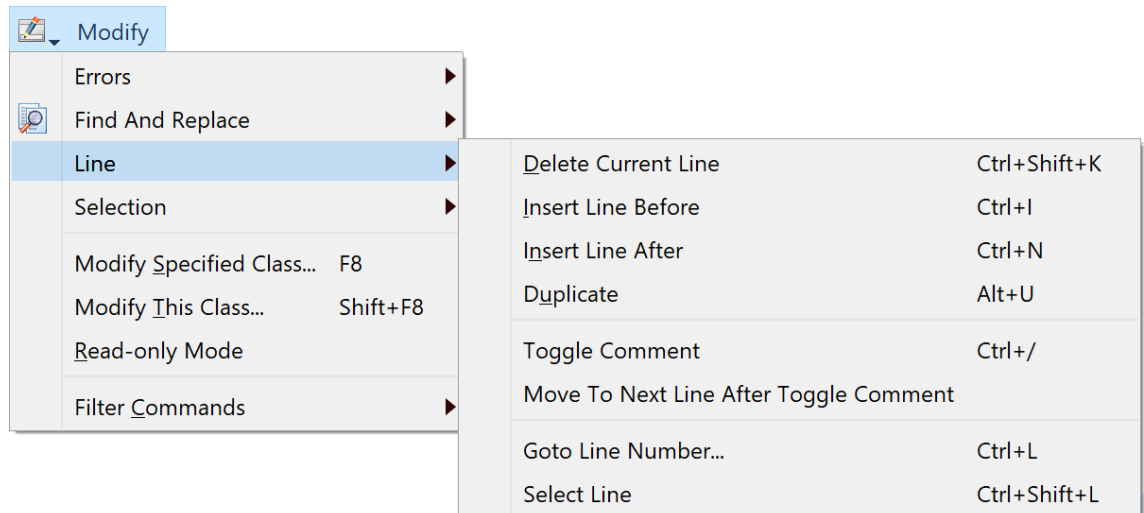
As you type characters into the find text field, the code text area dynamically updates to reflect the found text. It highlights the found text (e.g. the text 'lresponsedetails' is searched and highlighted in the above image), and it also adds a green marker to the vertical scrollbar, in a similar way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

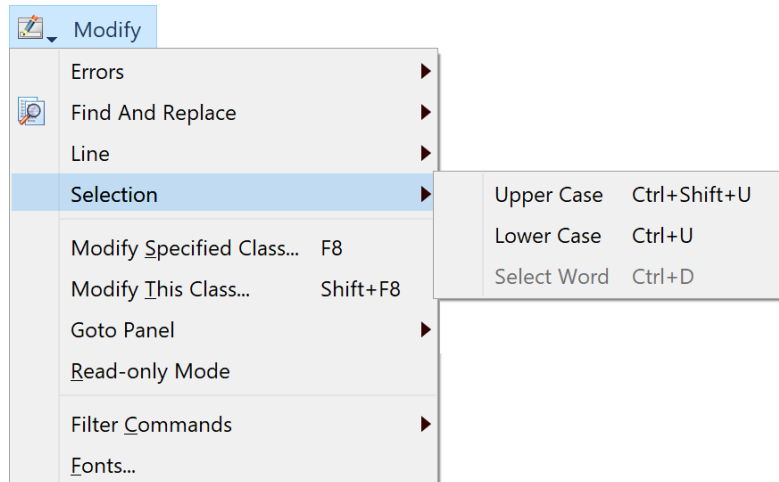
Commenting / Uncommenting Lines

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.



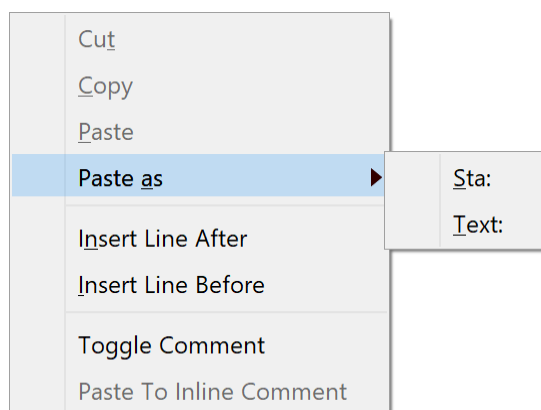
In addition, the Selection submenu contains the option **Select Word** which selects the word containing the insertion point, or where the insertion point is at the beginning or end of a word; in the latter case the word to the right or left of the insertion point is selected.

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 behavior.

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.



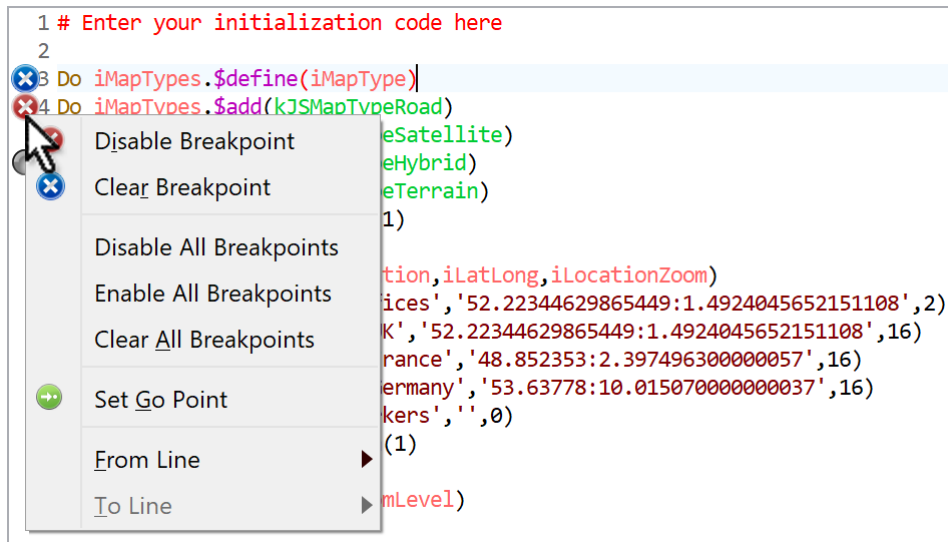
Setting Breakpoints & the Breakpoint Context Menu

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.

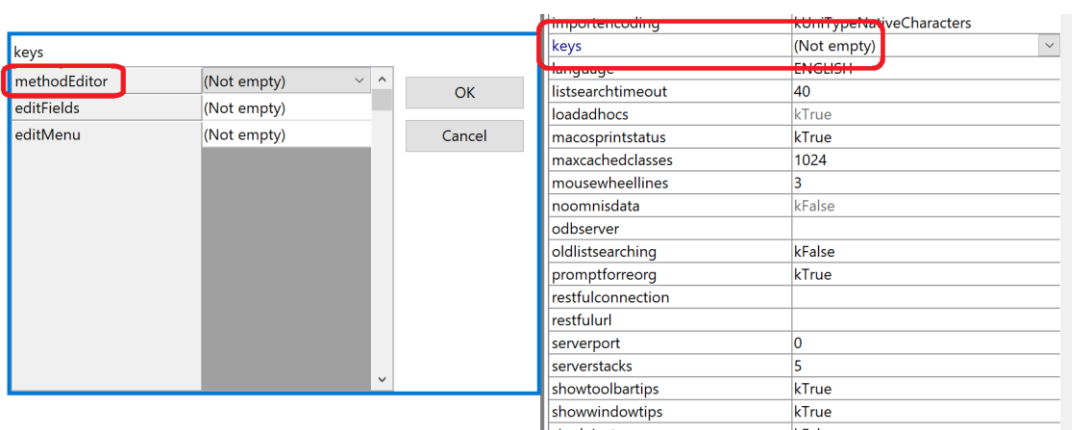
A complete and updated list of keyboard shortcuts has been added to the Studio 10.1 section earlier in this manual.

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 classvariablestyle	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

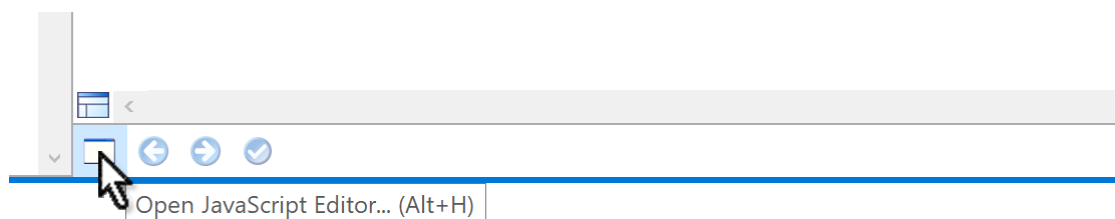
When you print methods using the File menu, Omnis now uses the syntax colors from the default theme (which is designed for a white background). You can turn off this behavior (and print everything using black text) by setting the entry “printMethodsWithSyntaxColors” in the “methodEditorAndRemoteDebugger”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

JavaScript: Editor

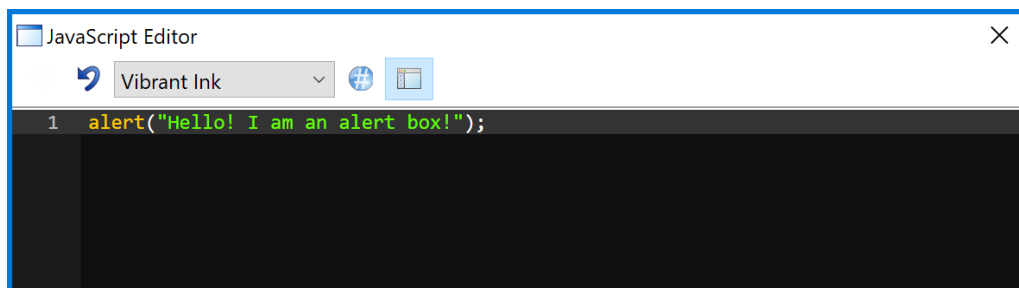
In addition to the main interface changes in the Method Editor, a JavaScript editor has been added to the Method Editor to allow you to enter a whole block of JavaScript code directly into the *JavaScript:* command, rather than line by line as in previous versions. The new JavaScript editor will popup 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
 - kEPllogStackAndReport:** 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
 - kEPllogStackAndContinue:** 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”.

The screenshot shows a form with four tabs: 'Basic Details', 'Career/Education Experience', 'Upload Supporting Files', and 'Submit Application'. The 'Basic Details' tab is active. Below the tabs, the text 'Please fill in the basic information below.' is followed by several fields: 'Title' (dropdown menu with 'Mr' selected), 'Date of Birth' (calendar widget for August 2018), 'First Name' (text input with 'Peter' and an orange highlight), 'Surname' (text input), and 'Gender' (radio button for 'Male').

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”.

The screenshot shows the same form with the 'Career/Education Experience' tab selected. The 'Career' and 'Education' sub-tabs are visible. Below the sub-tabs, the text 'Please tell us about your career history and education.' is followed by 'Employer Name' and 'Address' text input fields.

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 \$construct method of a remote form. The "omnisobject" <div> 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 <div> 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 behavior 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" <div>, but retain the default template.

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

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 configuration 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

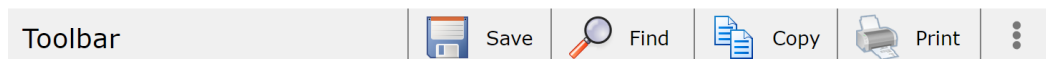
There are some new JavaScript controls, including a **Toolbar control** and an **iCalendar external object** (which can be used in remote forms and window classes), plus some of the existing JavaScript controls have been enhanced, including the **Segmented Control**, **Progress bar Control**, **File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

Toolbar Control

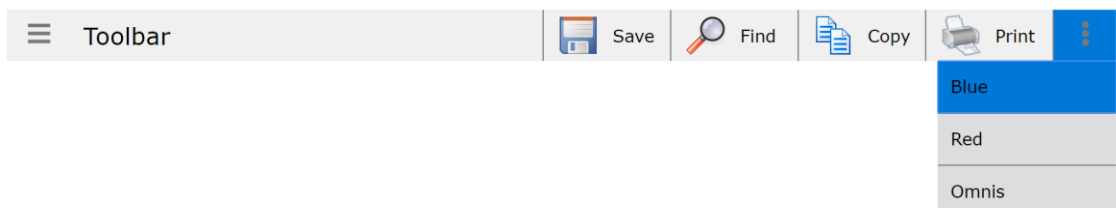
The **Toolbar Control** is a new JavaScript Control that can contain a number of buttons which the end user can click on or tap to perform an action. Each toolbar button can be assigned an icon and text, as well as a different action. When a button is clicked, the item number is reported to the event 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.

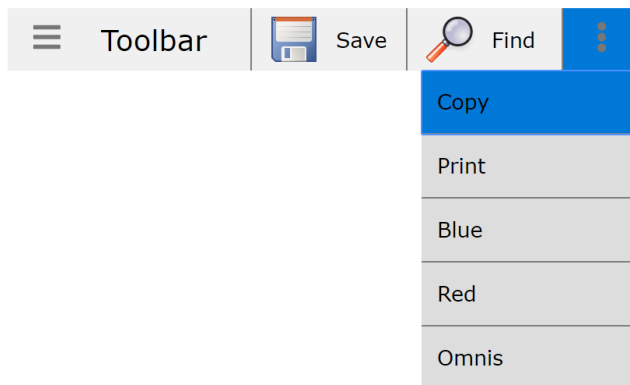
The following example Toolbar has four items or buttons defined, each with an icon and text, a main title 'Toolbar' on the left, and an overflow menu on the right.



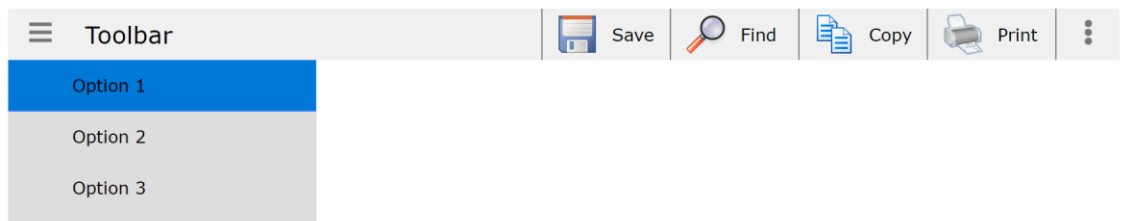
Items can be forced to always appear in the overflow, regardless of the width of the main toolbar, shown on the right of the toolbar by the three vertical dots, and shown dropped here:



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 <code>\$verticaltoolbar</code> property: Down or Up for horizontal toolbars, Right or Left for vertical toolbars
<code>\$toolbartitle</code>	The optional title to display on the toolbar; leave this blank for no title
<code>\$titlefontsize</code>	The font size applied to the toolbar title
<code>\$itemwidth</code>	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
<code>\$displaystyle</code>	A <code>kJSToolbarStyle...</code> constant determining the position of the icon text relative to the icon: either Above, Below, Left of, or Right of the icon
<code>\$showitemicons</code>	If true, any item with an <code>\$itemiconid</code> will display an icon on the toolbar; note that icons are not displayed on overflow items
<code>\$showitemtext</code>	If true, any item with <code>\$itemtext</code> will display text on the toolbar; when true, <code>\$showtooltips</code> is disabled
<code>\$showtooltips</code>	Show tooltips on toolbar items; <code>\$showitemtext</code> must be set to false
<code>\$showdividers</code>	If true, dividers will be shown between toolbar items
<code>\$dividercolor</code>	The color of the dividers between items if <code>\$showdividers</code> is true
<code>\$iconcolor</code>	The color of standard icons such as the hamburger icon
<code>\$sidemenucolor</code>	The background color of the side menu
<code>\$overflowcolor</code>	The background color of the overflow menu
<code>\$toolbaractivecolor</code>	The color of toolbar items when clicked
<code>\$toolbarhovercolor</code>	Hover color for toolbar items
<code>\$sidemenuhovercolor</code>	Hover color for side menu items
<code>\$overflowhovercolor</code>	Hover color for overflow items
<code>\$selecteditem</code>	The number of the selected item
<code>\$showselecteditem</code>	If true, the item will have its background color set to <code>\$selectedcolor</code> and its text colour set to <code>\$selectedtextcolor</code> .
<code>\$selectedcolor</code>	The background color of the selected item
<code>\$selectedtextcolor</code>	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 be 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()****OmnisCalendar.\$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()**OmnisCalendar.\$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 centered 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 <div> 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, '')
```

Product	Date Added	Color	Price	Available	Notes
Bag	21 Feb 2012		19.00	✓	
Balls	20 Feb 2012		4.55		Delivery r
Clubs	20 Dec 2011		299.99	✓	
Gloves	21 Feb 2012		19.99	✓	
Score Card	21 Feb 2012		19.99		Discontin
Tees	02 Feb 2012		17.59	✓	
Trolley	01 Jan 2012				
Umbrella	21 Feb 2012				
Bag	21 Feb 2012				
Balls	20 Feb 2012				Delivery r
Clubs	20 Dec 2011				
Gloves	21 Feb 2012				

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.

\$sprintcontents(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 center 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.

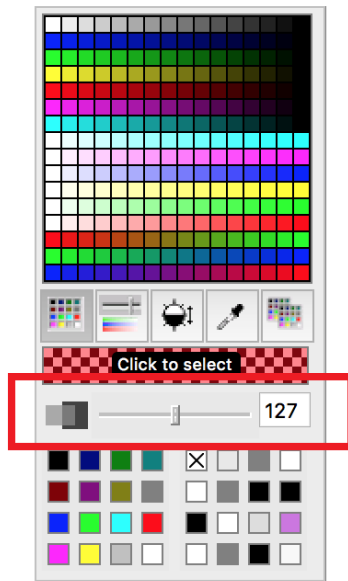
This property sets white-space for the control to 'normal' if true, and 'no-wrap' if false. Therefore, this may change the behavior if the HTML control is in a paged pane, and possibly in other cases where the white-space would have been inherited by a parent element.

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 RoundedRectangle 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 $cwind.$objs.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 `$currentnodeid` 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 behavior 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	<code>\$enabled</code>	<code>\$enabled</code>	<code>\$active</code>
Activity Control	<code>kTrue</code>	Removed	<code>kTrue</code>
Background Control	<code>kTrue</code>	Removed	<code>kTrue</code>
BarChart Control	<code>kTrue</code>	<code>kTrue</code>	<code>kTrue</code>

	Studio 8.x	Studio 10	
JS control	\$enabled	\$enabled	\$active
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
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

	Studio 8.x	Studio 10	
JS control	\$enabled	\$enabled	\$active
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
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, thick 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/RKJLekoR7T1Tv
    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 thick 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/jshtml/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`).

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

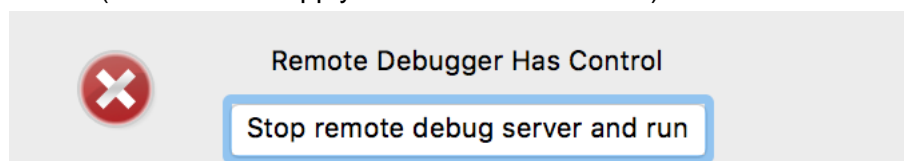
While the remote debugger is attached to a copy of Omnis, the local debugger is disabled in that copy. This also affects the ability to right click and view variable values.

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)
<code>\$keepconnectionopen</code>	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.
<code>\$requiresecureconnection</code>	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_addcgc	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_badhtmllesc	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 unicodev(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 configuration 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 `$::scale` property for the Gif external component has changed affecting how single- and multi-frame gifs are scaled. If `$::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 $obj.$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 `"oldMouseMoveBehindBehavior"` 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 `"oldWindowShadowBehavior"` 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

`$container` now returns the window instance for controls (including subwindows) at the top level: in previous versions this was not available for window class controls, only JavaScript controls. For example, you can use `$inst.$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 be 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 Studio 10.x: these commands were marked with “OBSOLETE COMMAND” in pre-Studio 10.x versions and appeared in the ‘Obsolete commands...’ group in the Command list (which is no longer available in the new Code Editor). 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 was not previously marked as obsolete but it has been removed from Studio 10.x and will be commented out in your converted code.

Autocommit OBSOLETE COMMAND	Enable receiving of Apple events OBSOLETE COMMAND
Begin SQL script OBSOLETE COMMAND	End SQL script OBSOLETE COMMAND
Build list from select table OBSOLETE COMMAND	Execute SQL script OBSOLETE COMMAND
Build list of event recipients OBSOLETE COMMAND	Fetch current row OBSOLETE COMMAND
Call method OBSOLETE COMMAND * (converted to Do code method)	Fetch first row OBSOLETE COMMAND
Cancel event recipient OBSOLETE COMMAND	Fetch last row OBSOLETE COMMAND
Cancel publisher OBSOLETE COMMAND	Fetch next row OBSOLETE COMMAND
Cancel subscriber OBSOLETE COMMAND	Fetch previous row OBSOLETE COMMAND
Close client import file OBSOLETE COMMAND	Get SQL script OBSOLETE COMMAND
Close cursor OBSOLETE COMMAND	Logoff from host OBSOLETE COMMAND
Commit current session OBSOLETE COMMAND	Logon to host OBSOLETE COMMAND
Declare cursor for OBSOLETE COMMAND	Make file class from server table OBSOLETE COMMAND
Delete client import file OBSOLETE COMMAND	Make schema from server table OBSOLETE COMMAND
Describe cursors OBSOLETE COMMAND	Map fields to host OBSOLETE COMMAND
Describe database OBSOLETE COMMAND	Open client import file OBSOLETE COMMAND
Describe results OBSOLETE COMMAND	Open cursor OBSOLETE COMMAND
Describe server table OBSOLETE COMMAND	Open desk accessory OBSOLETE COMMAND
Describe sessions OBSOLETE COMMAND	Perform SQL OBSOLETE COMMAND
Disable automatic publications OBSOLETE COMMAND	Prepare current cursor OBSOLETE COMMAND
Disable automatic subscriptions OBSOLETE COMMAND	Prompt for event recipient OBSOLETE COMMAND
Disable receiving of Apple events OBSOLETE COMMAND	Prompt for word server OBSOLETE COMMAND
Enable automatic publications OBSOLETE COMMAND	Publish field OBSOLETE COMMAND
Enable automatic subscriptions 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	Set hostname OBSOLETE COMMAND
Send core event OBSOLETE COMMAND	Set password OBSOLETE COMMAND
Send core event with return value OBSOLETE COMMAND	Set publisher options OBSOLETE COMMAND
Send database event OBSOLETE COMMAND	Set SQL blob preferences OBSOLETE COMMAND
Send finder event OBSOLETE COMMAND	Set SQL script OBSOLETE COMMAND
Send to publisher OBSOLETE COMMAND	Set SQL separators OBSOLETE COMMAND
Send word services event OBSOLETE COMMAND	Set subscriber options OBSOLETE COMMAND
Server specific keyword OBSOLETE COMMAND	Set transaction mode OBSOLETE COMMAND
Set batch size OBSOLETE COMMAND	Set username OBSOLETE COMMAND
Set character mapping OBSOLETE COMMAND	SQL: OBSOLETE COMMAND
Set client import file name OBSOLETE COMMAND	Start session OBSOLETE COMMAND
Set current cursor OBSOLETE COMMAND	Subscribe field OBSOLETE COMMAND
Set current session OBSOLETE COMMAND	Subscribe now OBSOLETE COMMAND
Set database version OBSOLETE COMMAND	Translate input/output
Set event recipient OBSOLETE COMMAND	Use event recipient OBSOLETE COMMAND