What's New in Omnis Studio 6.1

TigerLogic Corporation
December 2014
29-122014-02

Table of Contents

About This Manual

This document describes the new features and enhancements in Omnis Studio 6.1, 6.0.1, and 6.0 including many features in the JavaScript Client.

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

If you are new to Omnis Studio

If you are new to Omnis Studio you should start by reading the *Creating Web & Mobile Apps* manual and then the *Omnis Programming* manual. All the Omnis Studio manuals are available to download from the Omnis website at: www.tigerlogic.com/omnis.

When you start Omnis Studio you could take a look at the tutorial, sample apps, and product information available in the **Welcome screen** (available on the New Users button in the main Omnis toolbar). In particular, you should look at the **Sample Apps** available in the Welcome screen: you can open the apps in your web browser and examine the code and classes in each library. In addition, you should look at the **JavaScript Component Gallery** on the Omnis website which contains sample apps and code for most of the JavaScript components.

Software Support and Compatibility

The following release notes relate to Omnis Studio version 6.1, 6.0.1 and 6.0 and include information about software support and compatibility that you should be aware of before using Omnis Studio 6.x, or if you are upgrading from Omnis Studio 5.x or before.

Development Serial Numbers

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

Library and Datafile Conversion

Omnis Studio 6.1 will convert existing version 6.0.x and 5.x libraries – the conversion process is irreversible. Omnis Studio 6.1 will convert 5.x Omnis datafiles (note that non-Unicode datafiles will be converted to Unicode), but 6.0.x datafiles *will not* be converted in Omnis Studio 6.1.

IN ALL CASES, YOU SHOULD MAKE A SECURE BACKUP OF ALL OMNIS LIBRARIES AND DATAFILES BEFORE OPENING THEM IN OMNIS STUDIO 6.1.

Omnis Server Configuration

The installer for the Omnis App Server now allows you to enter the configuration settings for the Omnis Server including the settings for the existing WSDL-based Web Services and the new REST-based Web Services.

Note these are the same settings accessed via the **Server Configuration** option in the File menu in the Omnis Server. See also information about the new Omnis Server configuration file (config.json) described later in this document.

64-bit Omnis

The Omnis executable has been re-engineered to run on 64-bit processors but will be available for selected product configurations and platforms only. The following table summarizes the availability of the 64-bit version of Omnis Studio for selected products.

64-bit

Product	Windows	OSX	Linux
Omnis Studio 6.1 SDK	**YES**	NO	NO
Omnis App Server	**YES**	NO	**YES**
Omnis Runtime (CAL)	**YES**	NO	**YES**

32-bit

The 32-bit versions of Omnis Studio, including the SDK, App Server, and Runtime will continue to be provided for Windows and OSX, but not for Linux.

64-bit DAMs

The Omnis DAMs provided with Omnis Studio 6.1 use 64-bit architecture. This means that you will need to install separate 64-bit clientware where appropriate. The 64-bit DAMs are not interoperable with 32-bit client libraries and vice-versa. For single-tier and embedded DAMs, including DAMSQLITE, DAMOMSQL, DAMMYSQL, DAMPGSQL and DAMAZON, all necessary changes have been made. The 64-bit ODBC DAM requires the 64-bit ODBC Administrator library and should be used with 64-bit ODBC Drivers to ensure compatibility.

For further details on the DAMs and clientware configuration, please refer to the Omnis website: www.tigerlogic.com/tigerlogic/omnis/dams

Java on 64-bit

When using Java in the 64-bit version of Omnis you must define the environment variable OMNISJVM64 and set it to the location of your JVM library (and any other libraries required to run Java), such as under Windows:

```
C:\Program Files\Java\jre1.8.0_25\bin\server\jvm.dll
```

Product Activation

When you start Omnis Studio 6.1 for the first time, product activation will occur automatically. Product activation will occur in the Omnis Studio 6.1 SDK, Client Access Licenses (CALs), and Evaluation versions. The enhanced Product Activation brings greater end user installation controls designed to prevent illegal copying and sharing of both Omnis Studio and your software application licenses. In addition, Omnis Studio 6.x SDKs can be installed on up to five (5) licensed systems per developer to extend and support cross-platform development across a wider range of devices and environments. Note: The Exclusive Single User Runtime (ESU CAL) license is not subject to Product Activation.

You can use the function sys(227) in your code to return the hardware ID of the end user's computer.

Please see the 'Omnis Activation FAQ' at www.tigerlogic.com/omnis for full details about Omnis product activation, including information about how you can switch your license from one computer to another.

Windows and OS X Support

Omnis Studio 6.x will run under Windows 8 and OS X 10.9 (Mavericks). The minimum requirement for Omnis Studio 6.x to run under OS X is version 10.7 (Lion).

The functions iswindows8() and ismountainlion() have been added to allow you to detect these operating systems.

Windows 8.1 support

Omnis Studio 6.1 will run under Windows 8.1. All versions of Omnis Studio prior to this including Omnis Studio 6.0.x and 5.x and before are not supported under Windows 8.1.

OS X 10.10 support

Omnis Studio 6.1 will run on OS X 10.10 (Yosemite). All versions of Omnis Studio prior to this including Omnis Studio 6.0.x and 5.x and before are not supported on OS X 10.10.

PPC support

Omnis Studio 6.x will not run on Mac PPC.

Windows XP

Omnis Studio 6.x is not supported on Windows XP, 2000 or earlier. Specifically the MySQL DAM is no longer supported on these platforms.

Linux Support

There is no 32-bit Development, Runtime, or Server version of Omnis Studio 6.1 for Linux. However, the 64-bit Omnis App Server is available for Linux to allow you to run your web and mobile apps in a Linux server environment.

HAL and OpenSSL

Omnis Studio 6.1 for Linux no longer depends on HAL or OpenSSL, as in previous versions.

Windows Mobile

Support for the Windows Mobile Client has been removed from Omnis Studio 6.x. We recommend that you migrate all Windows Mobile based apps to the new JavaScript Client which is supported in web browsers on all Windows Phone and Windows 8 based computers and devices, including smartphones, tablets, and desktop computers.

Web Client Plug-in

The Web Client plug-in functionality has been hidden or disabled in the IDE by the removal of the components in the Webcomp folder, but the plug-in can be reinstated if required.

The files that enable support for the Omnis Web Client plug-in have been removed from the development version of Omnis Studio 6.x in order to remove the Web Client functionality from the IDE. This includes the **Web Client Components,** previously found in the Component Store (located in the 'Webcomp' folder in the Omnis tree), various tools or wizards in the IDE, and other associated files.

If you wish to use the Omnis Web Client plug-in functionality in your applications you need to download the Web Client Development kit installer from the Omnis website (www.tigerlogic.com/omnis). This installer contains all the necessary files to reinstate the Web Client components and functionality.

Web Client plug-in migration tool

Omnis Studio 6.x includes a tool (also present in Studio 5.2.x) to allow you to migrate remote forms that use the Web Client plug-in to the new JavaScript Client. In order to use the migration tool, you will need to reinstate the Web Client plug-in functionality and

components (available to download from the Omnis website). See the appendix at the end of this manual about the migration tool and the limitations of the migration process.

Java Beans

The Java Beans external and all supporting files have been removed from Omnis Studio 6.x.

Character External Functions

The following external functions (contained in xcharfnc.dll) have been removed from Omnis Studio 6.x: ansicode(), ansichar(), oemcode(), oemchar(), omniscode(), omnischar().

Pre-V30 SQL Functions

The Pre-V30 SQL functions insertnames(), selectnames(), updatenames(), and wherenames() have been removed from the Catalog (F9/Cmnd-9) in Omnis Studio 6.x, but they can still be used in your code. In addition, the createnames() and server() functions have been removed from Omnis, including the Catalog, and will no longer work in code in Omnis Studio 5.x/6.x onwards.

OLE2

OLE2 has been removed from Studio 6.1 by removing the OLE2 dll from the xcomp folder in the main Omnis product tree. You can download the OLE2 dll from the Omnis website if you wish to restore support for OLE.

What's New in Omnis Studio 6.1

Omnis Studio 6.1 provides several enhancements in the JavaScript Client technology to make the creation of web and mobile applications easier and quicker. In addition, Omnis Studio 6.1 provides support for RESTful based Web Services, and includes a 64-bit version of Omnis for some platforms and product configurations. The Omnis Studio 6.1 release includes the following features and enhancements:

- **JavaScript Components**
  There are a number of new JavaScript controls that have a "native" appearance on the platform on which they are running. Their style is defined in CSS and adapts for each client platform. These new JavaScript components are in a new group in the Component Store. In addition, the scripts used in the JavaScript Client have been optimized to improve overall performance
- **Web Services**
  Support for RESTful Web Services for client and server: you can create a user interface for RESTful web services, or expose your Omnis code on the Omnis App Server as a Web Service using the new component. In addition, there is a new JSON external component, called OJSON, that allows JSON based objects returned from RESTful resources to be manipulated
- **64-bit Omnis App Server**
  The Omnis executable has been re-engineered to run on 64-bit processors, while the 32-bit versions of Omnis Studio will continue to be provided. The 64-bit version of Omnis will only be available for selected product configurations and platforms. See the *Software Support and Compatibility* section for more details
- **Sync Screens Tool**
  There is a new tool that allows you to automatically configure the components on the different layouts stored in a single JavaScript remote form. This will save you a lot of time when designing forms and make your app more consistent for end users
- **Pie and Bar Charts**
  The JavaScript Pie and Bar Chart components have been enhanced and now include the ability to add axis titles and change the position of the legend
- **Transform Component**
  a new 'non-visual' external component that lets you animate or move objects on standard window classes (note this is not for JavaScript objects on a remote form)
- **Print Preview**
  There is a new Print Preview window that allows the end user to select text from the screen and review pages in a page list in the margin of the preview window

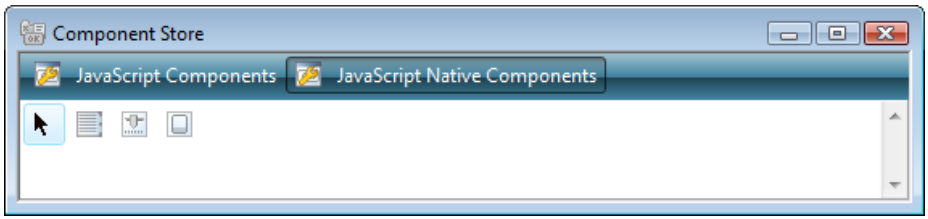
- **Max number of method lines**
  the max number of lines in an Omnis method has been increased from 1024 lines to 256,000
- **Comparing variables**
  you can now compare binary variables, object variables and object reference variables in your code – variables each side of the operator must be the same type
- **Multi-threaded separators**
  each thread on a multi-threaded Omnis App Server can have its own values for decimal point, thousands, and import decimal place indicator stored in $separators, allowing you to deploy multi-language mobile apps from the same Omnis library
- **Object references auto delete**
  Object references are now deleted automatically when they are no longer required in order to free up memory
- **Synchronization Server**
  Enhancements in the Omnis Synchronization Server include support for automatic device recognition by hardware-id, up to 255 user groups can be specified with 65535 unique devices per group, server-side replication, and the ability to upload or download only during synchronization

JavaScript Remote Forms and Components

Native JavaScript Components

There is a new group of JavaScript Components that have a more familiar or "native" appearance when they are displayed on different mobile platforms – the new appearance is rendered in the JavaScript Client using CSS customized for each platform. The different appearance for each platform is handled by Omnis automatically, therefore you only need to setup the component once in design mode.

The native components appear in a new group in the Component Store called 'JavaScript Native Components' – the existing JavaScript Components are still provided and are located in the 'JavaScript Components' folder. The Native components currently include the List, Slider, and Switch.



When running on a supported device, these new controls will render and work in a manner close to a device's native versions. For example, a native Switch control will look like an iOS switch on an iOS device, an Android switch on an Android device, and so on, while using a single control in design mode on your remote form.

JavaScript Remote forms now have an Appearance property called $defaultappearance which determines both how a native control is displayed in the design window, and how it would render on non-supported clients (e.g. Desktop browsers). The property can be set to one of the kAppearance… constants including: kAppearanceiOS, kAppearanceAndroid, and kAppearanceBlackberry.

The $defaultappearance property can also be switched using the new 'Native Components Display As' context menu option of a JavaScript Remote Form (right-click on the form to open the context menu). You can also cycle through the values using the keyboard shortcut Ctrl-Shift-N on Windows or Cmd-Shift-N on OSX when the remote form is the top window.

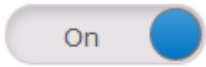
It is recommended that you set $disablesystemfocus property to kTrue for any native controls you have used, to prevent the focus rings being drawn around the controls when they are selected – otherwise the focus ring may interfere with the native appearance.

Native Switch

The Native Switch works, for the most part, in the same way as the standard JavaScript Switch component.

iOS

Android



BlackBerry

The Switch has a **$dataname** property, to which you can assign a **Boolean** instance variable. This will be kept up to date with the state of the switch.

The Switch has **$justifyhoriz** and **&justifyvert** properties. For some platforms (e.g. iOS) the switch maintains a particular aspect ratio. These properties determine how the switch is positioned inside the control in these circumstances.

Native Slider

The Native Slider works, for the most part, in the same was as the standard JavaScript Slider component.

iOS

Android

BlackBerry

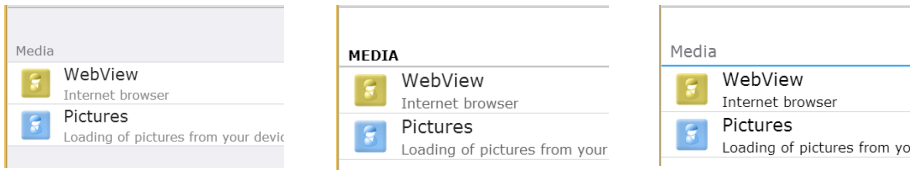
The current value of the slider is reported in the property $val according to where the slider is positioned. You can specify the range for the slider in the $::min and $::max properties. The $usessteps property is a Boolean determining whether or not the slider should snap to discrete step values specified in $step.

The Slider reports three events: evStartSlider (when the control is starting to track), evEndSlider (when the control has finished tracking), and evNewValue (when the value has changed). You can detect these events in the $event() method for the component. These events all pass the current value of the Slider in the pSliderValue parameter. As the user drags the Slider thumb the evNewValue event is triggered and pSliderValue is sent to the $event() method for the Slider.

If you do use evNewValue, you should mark your $event method as client-executed and consider enabling the $usessteps property and setting $step to limit the number of events being triggered as the user moves the slider. Alternatively, you could use the evEndSlider event to report the final value since for most purposes this will be the value selected by the user.

Native List

There are fewer real appearance differences between the different platform versions of the JavaScript Native List, as each platform allows you to customize its native list views to an extent which means there is no definitive list appearance.



iOS Android BlackBerry

In keeping with this philosophy, the JavaScript Native List exposes many appearance properties to allow you to customize the list how you wish. If you leave any of the values as kColorDefault, they will revert to the platform-specific default.

The native list has been designed along the same lines as the iTableView component of iOS forms. The list can either be a standard flat list or a grouped list, using nested lists.

To increase efficiency, the list only draws the displayed rows, along with several more in a buffer zone around them, at any one time. This provides smooth scrolling, and means that the size of the list has very little impact on performance. As a side-effect of this, when scrolling quickly, you will see that the rows may not be rendered immediately.

Defining the $dataname list

The structure of the List instance variable used for the $dataname of the native List control differs based on how you wish the list to display.

The **Data** tab in the Property Manager allows you to assign column numbers for each row part, i.e. $text1col allows you to specify which column in your list contains the the data to display as the main text of the row. If you do not wish to make use of a particular row part, leave its column set to 0.

You need to define and populate your list in accordance with the column numbers you have set in the **Data** tab of the Property Manager. The content of these columns should be as follows:

- **text1col:** This should be Character data, to display as the main text for the row.
- **text2col:** This should be Character data, to display as the secondary text for the row.
- **imagecol:** This should be Character data - a URL to an image to display.
  The image will be scaled to fit the size of the row's image (customized using $imageheight & $imagewidth).
  The URL can make use of Omnis' support for pixel-density-aware image selection by passing the URL in the format: "<URL to 1x image>;<Name of 1.5x image>;<Name of 2x image>" (where all 3 images are in the same location). This means that on a Retina device it will use the 2x image, but on a standard display device it will use the 1x image. As the image is scaled anyway, you could just always use the 2x image, but this method reduces unnecessary bandwidth usage and processing of larger images

- **accessorytypecol:** This should be a kJSNativeListAccessoryType… integer value. It determines the type of accessory to display on the right edge of the row.
  Use a value of 0 for no accessory. A *kJSNativeListAccessoryTypeNone* constant will be added in the future.
- **accessoryvaluecol:** Contains the current value of the row's accessory.
  This is currently only used by the Checklist accessory, to represent the checkbox's state
- **accessorycontentcol:** This should be Character data, and should describe the content for some accesssory types. The Accessory types which make use of this are:
  Button: For rows with a button accessory, the content should be the text for the button
  Custom: For rows with a custom accessory, this should be HTML to describe the contents of your custom accessory

Creating a Grouped list

If you wish your Native List control to display its data as a Grouped list, you need to change the structure of the list assigned to *$dataname*. The main list should comprise two columns:

- **Column 1:** Should be defined as being of type "List", and each row should contain a list structured as defined above (adhering to the columns specified in the *Data*tab of the property inspector). All of the rows defined in this sub-list become part of a single group.
- **Column 2:** A Character column, defining the name of the group.

Properties

The Native List component has many properties to allow you to customize its appearance. Most of these are self-explanatory and/or are described by their Property Manager tooltips. The following may need futher explanation:

- **$rowdisplaystyle:** Determines how the row is displayed. Value should be a *kJSNativeListDisplayXXX* constant. Current values allow you to change between displaying the two text fields in a vertical or horizontal fashion.

Advanced Customization

If you wish to alter the default appearance for a particular platform, or wish to change something which is not exposed as a property, you would need to do so using CSS. This is only recommended if you have experience of customizing CSS.

The classes used (when defaulting to platform-specific appearances) are defined in **native_list.css**.

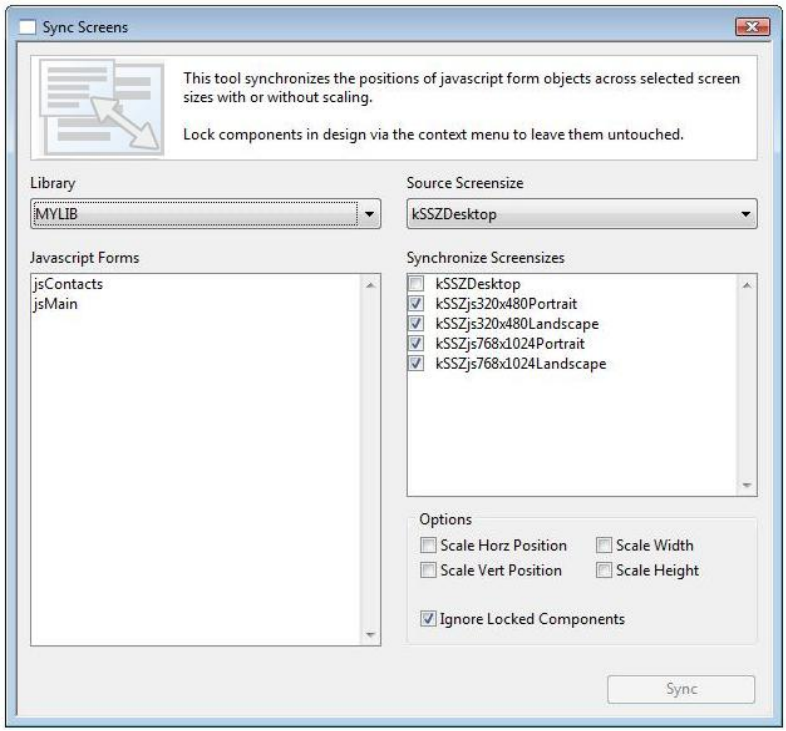
Rather than altering these classes here, it is recommended that you extend them in your **user.css** file, to prevent you needing to make these changes each time you update Omnis.

As always, when editing the CSS used by Omnis' controls, there is the possibility that you may change how a control appears or behaves (especially if you alter sizes), so you do so at your own risk.

Sync Screens Tool

When you create a remote form you can format all the fields and other controls on the form to appear the correct size and position for all possible orientations and screen sizes for desktop screens, tablets and phones (corresponding to different settings for $screensize). This is quite a time consuming task due to the number of different devices and layouts supported in each remote form.

In this release, there is a tool that automatically configures the components on the different layouts stored in a single remote form, which may save you a lot of time and make your apps more consistent and easier to use for end users across different devices. The new **Sync Screens** tool is available under the Tools>>Add Ons menu in the main Omnis menu bar.



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

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

Changing Remote Form Size

You can change the screen size ($screensize) for a JavaScript remote form using the design mode context menu on the remote form (right-click on the form to open the context menu). As a shortcut, when the remote form is the current top design window, you can use Ctrl-N/Cmd-N to cycle through the screen sizes for the remote form.

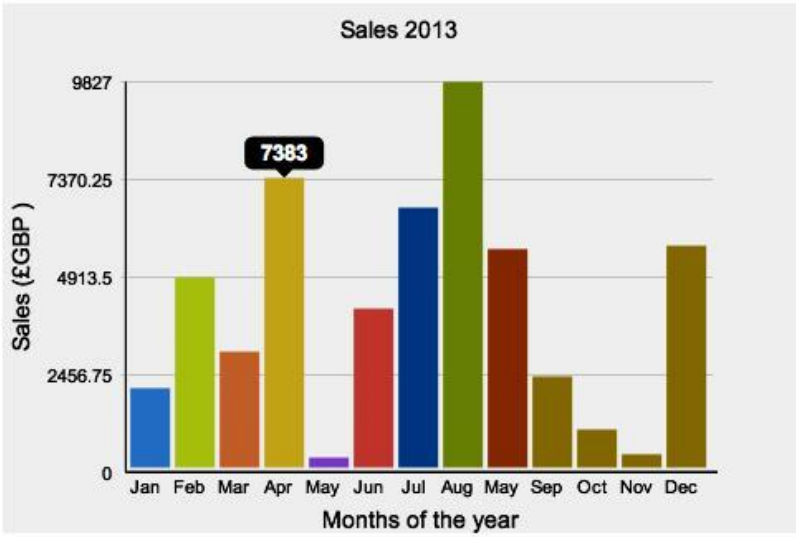
Pie and Bar Charts

The JavaScript Pie and Bar Chart components have been enhanced and now include the ability to add axis titles and change the position of the legend.

Bar charts

A number of properties have been added to the Bar Chart component to allow you to add a main title for the chart, as well as titles for the x and y axis. In addition, there are new properties to hide or show the x and y axis details or units.

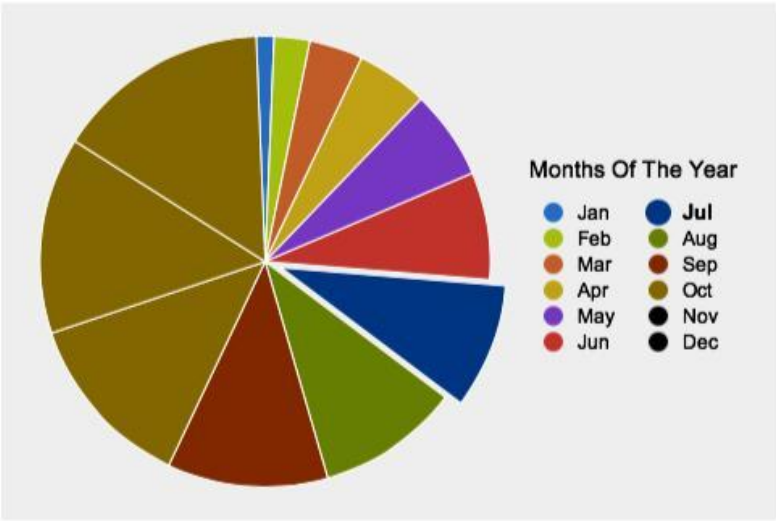
$maintitle	The main title for the chart
$xtitle	title for the x axis
$ytitle	title for the y axis
$showxaxis	if kTrue the chart shows x-axis details
$showyaxis	if kTrue the chart shows y-axis details



Pie charts

A number of properties have been added to the Pie Chart component to allow you to add a legend title and have more control over the appearance and positioning of the legend.

$maintitle	The legend title
$showlegendnames	If true the legend shows the value names (column 2) and not values (column 1). The list data structure is same as bar chart
$legendcolumns	The number of columns the legend is split into



Two new positioning constants have been added for the $legendpos property (which in practice are only appropriate for mobile devices), whereby as the device is rotated and the screen orientation changes, the legend is repositioned automatically either before (left/above) the pie or after (right/below) the pie. The new values are:

kJSPieLegendAutoBefore	the legend is placed before the pie chart, either above or to the left of the chart
kJSPieLegendAutoAfter	the legend is placed after the pie chart, either below or to the right of the chart

Selecting Icons

The 'Select an icon' dialog has been enhanced making it easier to browse and select an icon. Specifically, the dialog lists any custom icon sets you have added to the html/icons folder including any high definition (HD) icons (such as the 'myicons' icon set in the image below), the built-in icon data files (Omnispic, etc), the #ICONS system table containing the icons stored in the library, the icon pages within each icon set or data file, and the icon sizes.

See the 'Component Icons' section under 'What's New in Omnis Studio 6.0' for more details about creating and implementing your own HD icons for JavaScript components.

Passing Parameters to a Remote Task

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

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

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

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

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

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

Map Marker Icons

The marker list used to define the markers on the JavaScript Map Control (specified in $mapmarkers) can now have an optional fifth column which you can use to specify the icon URL for an image for the map marker. An empty string in this column (or a missing column altogether) means that the marker will use the default marker icon. The value for the marker icon is an icon URL which you set using the iconurl() function. Since the marker image has to be set for each row in your list you can specify a different image for each marker in the marker list, but if you want the same image for each map marker you still have to set the marker image for every row in your marker list.

Image Scaling and Alignment

The JavaScript Picture control has a new property to control the scaling and alignment of images within the control. The new property $keepaspectratiomode controls how the image in the control is scaled and positioned when $keepaspectratio is true and $noscale is false. The property value is a kKAR... constant with the possible values:

- **kKARtopLeft**
  The image is scaled to fit the control and anchored at the top-left corner. This is the default value and maintains compatibility with existing libraries
- **kKARcenter**
  The image is scaled to fit the control and centered, so background may be visible at the top and bottom or the left and right of the image, depending on the shape of the image control and the orientation of the image
- **kKARfill**
  The image is scaled to fill the control and centered, so no margin (background) is shown. This can result in the image being cropped at the top and bottom, or the left and right, depending on the shape of the image control and the orientation of the image

HTML Button Text

There is a new property in the JavaScript Button Control and TransButton control to allow you to add HTML styling to the button text. If true, $textishtml specifies that the text for the button (entered in $text) is treated as HTML, therefore any HTML can be used to style the text. For example, you can insert a line break by setting this new property to kTrue, and using
 in $text for the button wherever a line break is required.

The $textishtml property also allows other styling of the button text using various character and color attributes. Note that design mode does not render the HTML (the raw HTML code is displayed), and if you use attributes in the HTML they must be enclosed in single quotes.

Data Grid Rows

There is a new property for Data grids, $setlineheight, which allows you to center text vertically in the rows in the data grid. If true, the grid sets the line height so that text is vertically centered in each row (the default is kFalse).

Assigning Events to Multiple Objects

You can now select multiple objects of the same type and specify the events for all of the objects at the same time. For example, you can select a number of check boxes and enable the evClick event under the $events property to enable the event for all the selected check boxes.

Object name in Client Methods

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

Client Method Error Checking

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

Group Boxes

A Group box component is not available in the JavaScript client components, but you can create one "on the fly" using a new method called $makegroupbox() to change a Paged pane into something that simulates the behavior and appearance of a group box. The method PagePaneName.$makegroupbox(cLabel[,cFont,cFontSize,cTextColor]) must be executed on the client, and converts a Paged pane to a Group box with the specified label, as well as the optional CSS font, font size, and text color. This method can be called from $init in the remote form.

Drag Border Event

There is a new event for JavaScript fields, called evDragBorder, that is sent to one of the fields that share the border area being dragged. When it is triggered it could mean that the end user has resized the field (and therefore other fields in the same parent have resized) using the drag border. This event is implemented for JavaScript Client remote form controls, but also for window class fields.

Data Grid Columns

In order for the JavaScript Data grid to cater for multiple screen sizes, this component has a new property $columnwidthsarepercentage. If true, the column widths in the data grid specify a percentage of the width of the control rather than a specific number of pixels. This affects the properties $columnwidth, $::columnwidths, and $columnminwidth.

Rich Text Editor

There is a new property in the Rich Text Editor component to allow you to retrieve the plain text of the content in the control, that is, just the text without any HTML formatting. The $plaintextname property specifies the name of the variable that automatically receives the plain text equivalent of the data stored in the $dataname of the control. This property only needs to be set if you want the plain text value.

Tab Control

There is a new property for JavaScript Tab control called $canclickselectedtab, located on the Action tab of the Property Manager. If true, a click on the selected tab generates evTabSelected (provided that evTabSelected is specified in $events). This allows you to detect a click on the currently selected tab which was not possible in previous versions.

JavaScript Client Performance

The scripts used in the JavaScript Client have been optimized to improve overall performance.

Application Wrappers

Remote Tasks and the Wrappers

The Application Wrappers now send a unique device ID when connecting to the Omnis App Server. Omnis checks whether there is already a remote task instance with the same device ID and form name in the current library, and if it finds one, it will close it before opening a new task instance. This means that wrappers will not free the remote task connection when they timeout (as you cannot trap this event), but when the app is re-opened, it will close the old task before opening a new one.

Floating Controls

The SettingsFloatControls option in the iOS wrapper config.xml can now be set to 1 to cause controls to float in the iOS wrapper. This is useful when developing forms that use

the $screensize setting kSSZjs320x480Portrait and will run on iOS based mobile devices that are larger than iPhone 4 (320x480), such as the iPhone 5 (320x568).

iOS Wrapper Licensing

The iOS application wrapper uses the UICKeyChainStore wrapper, created by kishikawakatsumi and governed by the MIT license: more info is available here: https://github.com/kishikawakatsumi/UICKeyChainStore

If you distribute your Omnis app using the iOS wrapper you will need to comply with the terms of this license and include the MIT requirements in your own software license.

Updating the SCAF

When you make changes to CSS files or scripts you must restart Omnis to allow the SCAF to be rebuilt to reflect the file changes, and therefore to ensure your app is built using an up-to-date SCAF. For examples, Omnis needs to be restarted after the 'user.css' has been changed to ensure the omnis.scaf contains the updated style sheet.

Web Services

There is a new Web Services component that provides support for RESTful web services for client and server. The new component instantiates a new Web Worker Object with properties and methods based on the type of web service you are using. You must create the client interface to the Web Service Object.

There is also a new Web Server plug-in to allow the Omnis App Server to expose your Omnis code as a RESTful Web Service. This is described under the *Creating your own Web Services* section.

In addition, there is a new JSON external component, called OJSON, that allows JSON based objects returned from RESTful resources to be manipulated: this is described under the *Manipulating JSON Resources* section below.

What is REST?

REST is the predominant architectural style being used today to consume and publish Web Services, and is seen as an alternative to other distributed-computing specifications such as SOAP. A RESTful Web Service is identified by a URI, and a client interacts with the resource via HTTP requests and responses using a fixed set of HTTP methods.

A RESTful Web Service follows the following rules to provide access to the resources represented by the server:

1. The resource must be identified by a URI, which is a string of characters similar to a web address, that points to the resource.
2. A client interacts with the resource via HTTP requests and responses using a fixed set of HTTP methods.
3. One or more representations of a resource can be returned and are identified by media types.
4. The content of a resource can link to further resources.

There are two sides to consider for RESTful Web Services:

- a **Client,** which would be an Omnis library containing methods to consume a RESTful Web Service,
- and the **Server,** where you can implement a RESTful Web Service by exposing the business logic (remote task methods) in your Omnis library to be consumed by clients.

Example Library

This is an example library that demonstrates the capability of Omnis to consume a RESTful web service – the library is available with a Tech Note: TNWS0002 which is available on the Omnis website: www.tigerlogic.com/omnis.

The example library presents basic weather forecast information by consuming a web service provided by WorldWeatherOnline.com. The company provides two versions of their API; a free version and a premium one. The example is based on the free API service which you can call up to 12,000 times a day and up to 5 times per second that provides enough data for the purposes of this example.

API Key

To use the web service consumed in the example library, you must obtain an API key from WorldWeatherOnline.com (which must not be shared with other people): note the key used by TigerLogic to create and test the example library has been removed. To obtain your own API key you need to go to www.worldweatheronline.com and select the Free Signup option (https://developer.worldweatheronline.com/auth/register), and after completing the signup, you can generate an API key at https://developer.worldweatheronline.com.

When you open the example Omnis library, if there is no API key present, you will be presented with a window to enter the key. This value is stored in the remote task and you should not be asked to enter it again.

If you reuse any portion of the example app for your own development and deployment, or create your own application using the information from World Weather Online, please remember to obtain an API key for your own or your clients use.

Testing the Example Library

To test the web service and display the weather forecast, open the example library, right click on the **jsWeather** remote form and select 'Test Form' from the context menu. The form should open in your desktop browser and show the current weather for Saxmundham (the home town for TigerLogic's Omnis development team in the UK) – the same information can be accessed in a table format by selecting the Table hyperlink. In the main remote form there is a pictorial summary of today's weather with the maximum & minimum expected temperatures, along with the forecast for the next four days. You can find the forecasts for other locations by entering either the city name or zip/post code.

The World tab gives a summary forecast for 20 selected cities in the world. Since the example library uses the free version of the API and there is limit of 5 queries per second, the world data is generated by batching the cities and using a timer to prevent a server error.

As the forecasts published by World Weather Online are only changed every 3 or 4 hours, all data is cached within the example library for 1 hour to prevent unnecessary calls back to the server.

If you publish the form to a webserver, when the form opens, it will try to identify your location using the IP address returned from the remote task. If this fails, it will revert to Saxmundham as the default location.

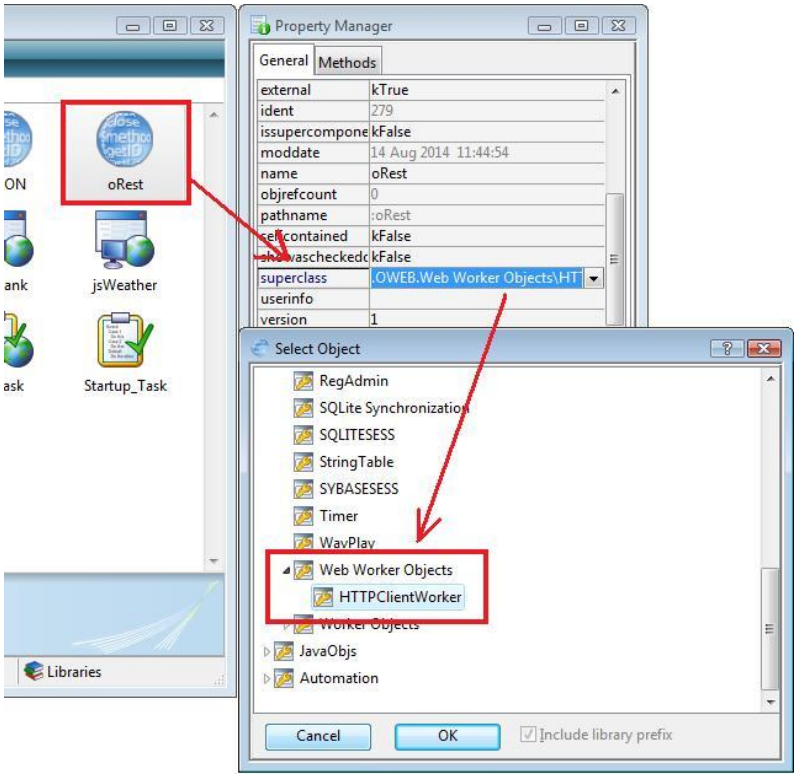
Creating a Web Services Client

There are a few key requirements for creating a Web Services Client which are:

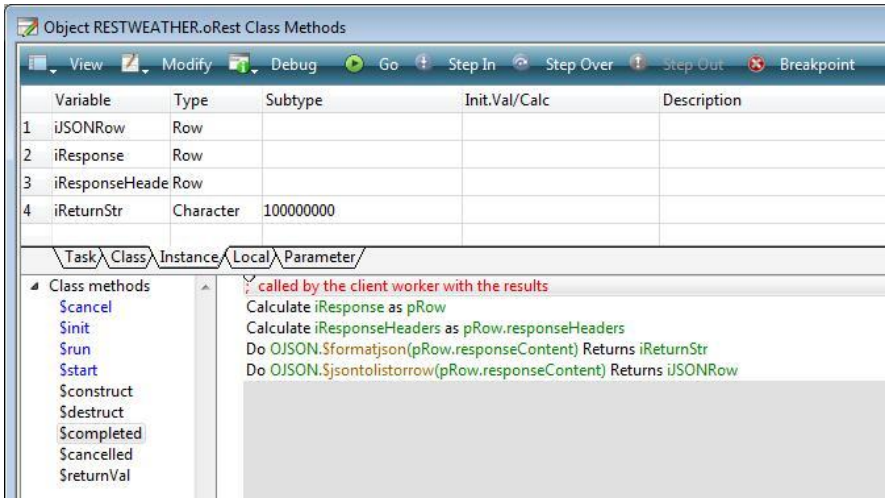
1. An HTTP client that allows resources to be submitted and received using various HTTP methods.
2. An HTTP client that allows HTTP headers to be specified for requests, and analysed for responses.
3. A means to manipulate the important media types for RESTful resources: XML and JSON.
4. Support for HTTPS.
5. Support for HTTP basic and digest authentication.

The new Web Services Client is implemented as a new External Component Object. The External Objects group in Omnis Studio now includes a group called **Web Worker Objects.** Currently this contains **HTTPClientWorker** which is an HTTP Worker Object. This worker object functions in a similar manner to the DAM worker objects, although there is a simplification in the way they handle re-use of the object when a request is currently in progress: see the notes about $init.

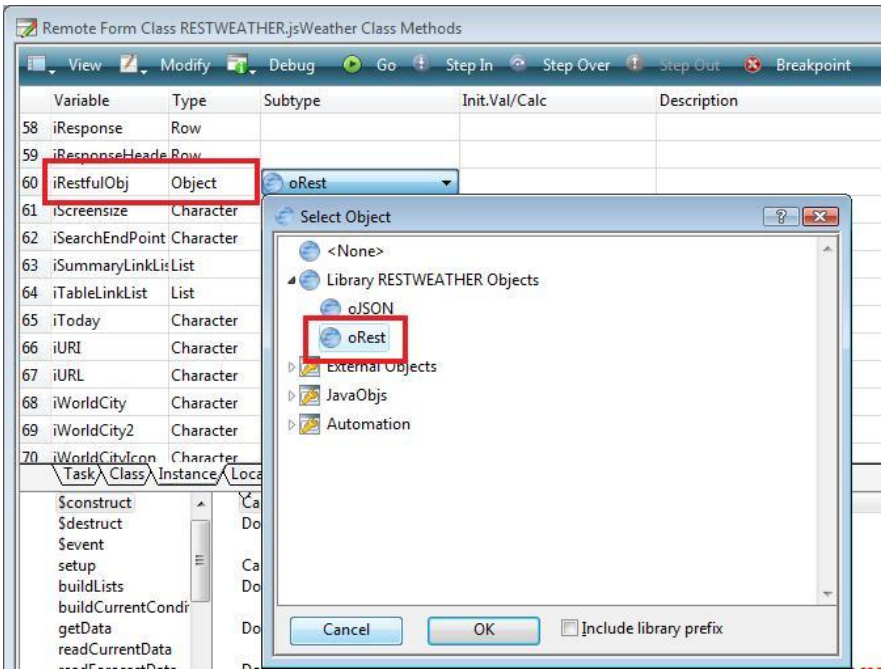
To use the HTTP worker object, you create an **Object Class** which is a subclass of HTTPClientWorker. Having created a new Object class, set its $superclass property to the name of the HTTPClientWorker object by clicking on the dropdown list and selecting the object in the Select Object dialog.



In the object class, you need to define two methods, $completed and $cancelled, which the client worker calls with either the results of a request, or to say the request was cancelled.



You then need to create an Object instance variable in your JavaScript remote form based on the new Object class to instantiate the object and allow you to interact with the web service by running its methods.



Properties

The HTTPClientWorker object has the following properties:

$state

A kWorkerState... constant that indicates the current state of the worker object.

$errorcode

Error code associated with the last action (zero means no error).

$errortext

Error text associated with the last action (empty means no error).

$threadcount

The number of active background threads for all instances of this type of worker object.

$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 $init for this object.

$shareconnections

If true (default) the object shares connections to HTTP servers with other HTTPClientWorker objects rather than managing its own set of connections (consider

authentication when setting this - different objects may need different authentication credentials, in which case you should not share connections). Must be set before executing $init for this object.

$timeout

The timeout in seconds for requests from this object. Zero means default to the standard timeout for the HTTP client. Must be set before executing $init for this object.

Methods

The HTTPClientWorker object has the following methods:

$init()

$init(cURI,iMethod,lHeaders,vContent[,iAuthType,cUserName,cPassword,cRealm])
Initializes the worker object so that it is ready to perform the specified HTTP request. Returns true if the worker was successfully initialized. The parameters are:

Parameter	Description
cURI	The URI of the resource, optionally including the URI scheme (http or https), e.g. http://www.myserver.com/myresource. If you omit the URI scheme, e.g. www.myserver.com/myresource, the URI scheme defaults to http
iMethod	A kOWEBhttpMethod... constant that identifies the HTTP method to perform: kOWEBhttpMethodDelete, kOWEBhttpMethodGet, kOWEBhttpMethodHead, kOWEBhttpMethodOptions, kOWEBhttpMethodPatch, kOWEBhttpMethodPost, kOWEBhttpMethodPut, kOWEBhttpMethodTrace
lHeaders	A two column list where each row is an HTTP header to add to the HTTP request. Column 1 is the HTTP header name, e.g. 'content-type' and column 2 is the HTTP header value, e.g. 'application/json'
vContent	A binary or character variable containing the content to send with the HTTP request. If you supply a character variable, the worker converts it to UTF-8 to send in the request. Content can only be sent with Patch, Post and Put methods
iAuthType	A kOWEBhttpAuthType... constant that specifies the type of authentication required for this request. If you omit this and the remaining parameters, there is no authentication (and this parameter defaults to kOWEBhttpAuthTypeNone). Supported values are kOWEBhttpAuthTypeNone, kOWEBhttpAuthTypeBasic and kOWEBhttpAuthTypeDigest
cUserName	The user name to use with authentication types kOWEBhttpAuthTypeBasic and kOWEBhttpAuthTypeDigest
cPassword	The password to use with authentication types kOWEBhttpAuthTypeBasic and kOWEBhttpAuthTypeDigest
cRealm	The realm to use with authentication type kOWEBhttpAuthTypeDigest

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

$run()

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

The following method is from the example library – the method initializes the Web Services object, having already setup the main parameters for the $init() method, and calls the web service.

```
; iURI (Char) initialized as "http://api.worldweatheronline.com"
; iHTTPMethod (Int) set to kOWEBhttpMethodGet
; iHeadersList (List)
; iContentChar (Char)
Do iHeadersList.$define(iHeaderName,iHeaderValue)
; call the web service
Do iRestfulObj.$init(iURI,iHTTPMethod,iHeadersList,iContentChar)
Do iRestfulObj.$run() Returns lStatus
```

See $completed for handling the response from the web service.

$start()

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

$cancel()

If required, cancels execution of the worker on the background thread. Will not return until the request has been cancelled.

Callbacks

The HTTPClientWorker object calls the following callbacks, which must be defined in your object class:

$cancelled

Called to report that the request has been cancelled.

$completed

Called to report completion of the request. It has a single row variable parameter with columns as follows, including the content returned in the final column:

Column	Description
errorCode	An integer error code indicating if the request was successful. Zero means success, i.e. the HTTP request was issued and response received - you also need to check the httpStatusCode to know if the HTTP request itself worked.
errorInfo	A text string providing information about the error if any.
httpStatusCode	A standard HTTP status code that indicates the result received from the HTTP server.
httpStatusText	The HTTP status text received from the HTTP server.
responseHeaders	A row containing the headers received in the response from the HTTP server. The header values are stored in columns of the row. The column name is the header name converted to lower case with any - (hyphen) characters removed, so for example the Content-Length header would have the column name contentlength.
responseContent	A binary column containing the content received from the server.

The following is the $completed method in the oRest object in the example library:

```
;  called by the client worker with the results
Calculate iResponse as pRow
Calculate iResponseHeaders as pRow.responseHeaders
Do OJSON.$formatjson(pRow.responseContent) Returns iReturnStr
Do OJSON.$jsontolistorrow(pRow.responseContent) Returns iJSONRow
```

The new JSON external component can be used to process the JSON output.

CA Certificates

For secure connections, the old web enabler commands use the CA certificates in the folder secure/cacerts of the Omnis tree to verify connections.

The HTTPClientWorker uses the same certificates, except they have been packaged into the file omnisTrustStore in the folder secure/cacerts. If you wish to add further certificates to this file that are specific to your application, you can do this using the Java keytool utility:

```
keytool -importcert -noprompt -file %1 -alias %2 -keystore
   omnisTrustStore -storepass xxxxxx
```

The password xxxxxx is always used for this file, as there is nothing private stored within it. To run the command, supply the pathname of the new certificate in %1 in the above command, and an alias for the certificate (just a unique name within the file) in %2. The command line above assumes secure/cacerts is the working directory.

Manipulating JSON Resources

There is a new JSON external component, called OJSON, that allows JSON based content to be manipulated by Omnis applications. JSON is a text form that allows you to transmit data objects consisting of attribute–value pairs, and is an alternative to XML.

The new JSON component provides two ways to generate and parse JSON objects and documents:

1. A basic mechanism that simply maps directly between an Omnis list or row and JSON. This uses static methods in the OJSON component.
2. A more structured mechanism that uses an external component object called JSON in the OJSON external component.

Data Structure and Addressing

JSON maps on to a hierarchical Omnis list or row, with one exception, namely that it allows for mixed types in arrays – Omnis can cater for mixed types in list columns internally, but there is no support in the Omnis language for such a list column. Therefore:

1. you can map a JSON object to an Omnis row variable, where the column names are the JSON object member names.
2. you can map a JSON array where all of the members have the same primitive type (or null) to a single column Omnis list.
3. you can map a JSON array where the members are arrays or objects, or where there is more than one primitive type, to an Omnis row variable with columns named___1, ___2, etc. Note that this restricts such arrays to a dimension of 32000 or less.

Note: Primitive types are string, integer, float and Boolean.

The external component JSON object implemented in OJSON uses a character string called the member id to address an entity in a JSON document (an entity is essentially a node in the JSON document tree, so it can represent a primitive type, null, an array or an object):

1. The member id of the root of the document is the empty character string.
2. The member id for other objects is the dot-separated path through the document to the entity e.g.

a. If the root is an object with members a, b and c, the member ids for the object members are: a b and c

b. If b is an array with two elements, the member ids of the array elements are b.[0] and b.[1]

c. If b.[1] is an object with string member c, then c has the member id b.[1].c

Using member ids you can directly address any entity in the JSON document. Note that the empty member id can only be used to address the array or object which the root of the JSON document tree.

Static Methods

The OJSON object provides the following static methods:

OJSON.$jsontolistorrow(vData,[&cErrorText])

Parses the JSON array or object in vData (either binary (UTF8/16/32) or character) and returns a row or a list representing the JSON. Returns NULL and cErrorText if an error occurs.

OJSON.$listorrowtojson(vListOrRow,[iEncoding=kUniTypeUTF8,&cErrorText])

Writes the list or row and returns JSON with the specified encoding (UTF8,UTF16BE/LE,UTF32BE/LE or Character). Returns NULL and cErrorText if an error occurs.

OJSON.$couldbearray(vData)

Returns true if vData (either binary (UTF8/16/32) or character) could possibly be a JSON array because its first character is [.

OJSON.$couldbeobject(vData)

Returns true if vData (either binary (UTF8/16/32) or character) could possibly be a JSON object because its first character is {.

OJSON.$formatjson(vData)

Parses the JSON in vData (either binary (UTF8/16/32) or character) and returns a formatted representation (or error message if parsing fails) suitable for use in a multi-line entry control.

JSON External Component Object

After constructing the OJSON object, it represents an empty JSON object. The methods supported by the external component object (with the exception of $getlasterror()) all set an error code and error text if an error occurs during their execution. In addition, you can use the method $runtimerrors to set a flag thatcauses the component to generate a runtime error (entering the debugger if applicable) when an error occurs - this can be useful when developing code that uses OJSON. The object provides the following methods:

Note: in all of these descriptions, cMember is the member id of an entity in the JSON document tree:

$getjson([cMember,iEncoding=kUniTypeUTF8])

Returns the JSON for the OJSON object (when cMember is an empty string or omitted) or the specified array or object member (cMember) using the specified encoding.

$setjson(cMember,vData)

Sets the OJSON object (when cMember is an empty string) or the specified array or object member (cMember) to the JSON supplied in vData (either binary (UTF8/16/32) or character). Returns a Boolean, true for success.

$getlasterror([&cErrorText])

Returns the error code from the last OJSON object method executed; also optionally populates cErrorText with a description of the error.If no error occurred, returns zero and the error text is empty.

$runtimeerrors(bGenerate)

Call with true to cause a runtime error when a method returns an error (so the debugger is entered if applicable), or false to stop runtime errors. Returns previous value of bGenerate. Default is no runtime errors.

$listmemberids()

Returns a single column list of the member ids for all of the members.

$isobject(cMember)

Tests specified member. Returns true if the member is a JSON object.

$getobject(cMember)

Gets the specified object. Returns a row containing the object members or NULL if the member is not an object.

$setobject(cMember,wRow)

Sets the specified member to the object specified in wRow. Returns a Boolean, true for success.

$addmember(cMember,cNewMemberName,vValue)

Adds member cNewMemberName with value vValue to object cMember. Returns a Boolean, true for success.

$removemember(cMember,cMemberName)

Removes member cMemberName from object cMember. Returns a Boolean, true for success.

$hasmember(cMember,cMemberName)

Returns true if cMemberName is a member of object cMember.

$listmembers(cMember)

Returns a single column list which contains the member names of the object cMember. Returns NULL if cMember is not an object.

$isarray(cMember)

Tests specified member. Returns true if the member is a JSON array.

$getarray(cMember)

Gets the specified array. Returns NULL if the member is not an array, a single column list if the array elements all have the same primitive type or are NULL, or for mixed arrays a row with one column per array element.

$getarraylength(cMember)

Returns the number of elements in the array cMember. Returns zero if cMember is not an array.

$setarray(cMember,vListOrRow)

Sets specified member to an array.Accepts either a single column list or a row where the columns are the array elements (the latter allows for arrays of mixed types). Returns a Boolean, true for success.

$push(cMember,vValue)

Adds an element with value vValue to the end of the array cMember. Returns a Boolean, true for success.

$pop(cMember)

Removes the last element from the end of the array cMember and returns its value. Returns NULL if cMember is not an array or if cMember is empty.

$isstring(cMember)

Tests specified member. Returns true if the member is a JSON string.

$getstring(cMember)

Gets specified string member. Returns JSON string value (empty if member is not a string). Unescapes JSON syntax.

$setstring(cMember,cString)

Sets specified member to JSON string with value cString. Returns a Boolean, true for success.

$isbool(cMember)

Tests specified member. Returns true if the member is a JSON Boolean.

$getbool(cMember)

Gets specified Boolean member. Returns Boolean corresponding to JSON Boolean (false if member is not a Boolean).

$setbool(cMember,bBool)

Sets specified member to JSON Boolean with value bBool. Returns a Boolean, true for success.

$isinteger(cMember)

Tests specified member. Returns true if the member is a JSON integer.

$getinteger(cMember)

Gets specified integer member. Returns integer 64 bit (zero if member is not integer).

$setinteger(cMember,iInt)

Sets specified member to JSON integer with value iInt. Returns a Boolean, true for success.

$isfloat(cMember)

Tests specified member. Returns true if the member is a JSON floating point value.

$getfloat(cMember)

Gets specified floating point member. Returns number floating dp (zero if member is not floating point).

$setfloat(cMember,nNum)

Sets specified member to JSON floating point with value nNum. Returns a Boolean, true for success.

$isnull(cMember)

Tests specified member. Returns true if it is null.

$setnull(cMember)

Sets the specified member to null. Returns a Boolean, true for success.

Creating your own Web Services

This version of Omnis introduces a new Web Server plug-in which allows you to expose the code in your Omnis applications and provide them as Web Services for any clients to consume. The interface for the web services you can create and provide to clients is exposed as an API or set of APIs. The key requirements for Omnis to act as a server or provider of RESTful based Web Services are:

1. Allow an HTTP client to submit and retrieve resources using various HTTPmethods.
2. Expose the HTTP headers that arrive with a request, and allow headers to bespecified for the response.
3. A means to manipulate the important media types for RESTful resources: XML and JSON.
4. Support for HTTPS, and for HTTP basic and digest authentication.

These requirements can be met with a combination of the Omnis App Server with the new Web Server plug-in and a standard HTTP Web Server.

Omnis RESTful APIs

The new functionality allows Omnis RESTful APIs (or ORAs) to be fully defined using a "Swagger" definition (see https://github.com/wordnik/swagger-spec/), which is the most widely used standard for defining RESTful APIs. The reasons for chosing swagger include:

- It makes it easier to document and test them
- It has tools to generate clients for various languages using the swagger definition
- It simplifies the generation of RESTful APIs in Omnis, so you can concentrate on application logic rather than lower level protocol related issues

Note however that there is nothing in the new implementation that requires a developer to use the Swagger definition in order to use the new implementation.

Web Services in the Omnis IDE

Omnis RESTful APIs are visible in the Studio Browser as children of the library node beneath the Web Service Server node (the same as previous implementations of Web Services based on WSDL files). Each ORA is shown a separate node icon in the tree, and various options or actions are shown as hyperlinks when an ORA is selected. Omnis RESTful APIs have a method list and an error log that you can use to manage the service.

Omnis RESTful APIs have Swagger definitions rather than WSDL files, and the Swagger definitions can be viewed using the IDE browser hyperlinks for the ORA. There is a hyperlink for the top-level resource listing, and a separate hyperlink for each top-level URI component. Clicking on a link displays the relevant Swagger data for the link (building it if necessary first). In the top of the panels is a read-only URL; you can select the text for the URL, and paste it into a browser or into Swagger UI (in the latter case, the resource listing URL is the only URL you would use). Note that you need to be aware of potential CORS issues when using these links in Swagger UI (see the later section on CORS).

Creating an Omnis RESTful API

To create a Web Service or Omnis RESTful API you need to set some properties of a remote task and add some RESTful methods. The remote task class has two new properties to allow you to setup the Web Service:

- **$restful**
  If true, the remote task is RESTful, it can have URI objects, and can be part of a RESTful API by setting $restfulapiname. This property can only be set to kFalse when the remote task and superclasses have no URI objects
- **$restfulapiname**
  If not empty, this is the name of the RESTful API in the library containing the remote task (cannot equal $webservice for remote tasks in lib). The RESTful API name in this property must start with an alphanumeric (a-z) and can only contain a-z, 0-9 and _

To create an Omnis RESTful API (ORA), set the $restful property of a remote task to kTrue, and provide a name in $restfulapiname. Note: the $restful property is an inherited property, so if you create a subclass of a remote task with $restful set to kTrue, the subclass will also be $restful. Further note a remote task with $restful set to kTrue is not yet a member of an ORA. For each remote task that is to belong to an ORA (meaning that it provides URIs and methods for clients to call) set $restfulapiname. Note that all remote tasks in an ORA must be in the same library.

After setting $restful and $restfulapiname for a remote task, the new ORA will not appear in the browser, because it has not implemented any RESTful methods. Therefore the next step is to open the method editor for the remote task, in order to add objects and methods.

When a remote task is RESTful, the remote task has a group of objects (named $objs as usual). These objects are the URIs exposed by the remote task to clients. Inheritance works with these objects and their methods in the same way that it works with other Omnis classes that support inheritance. However, $cfield and $cobj are not resolved for URI objects

URIs

A URI must have one or more components starting with /. Parameter place-holders can be included as component two or later as {paramName} where paramName is unique (case-insensitive) in the URI. The URI cannot have a trailing / and cannot be duplicated. For example:

- /users
- /user/{userId}

In the second case, userId is a parameter place-holder, meaning that the RESTful methods implemented for the URI must all have a parameter named userId which Omnis populates with the userId from the addressed URI.

A URI is considered to be a duplicate (and therefore not allowed in the remote task) if it has the same number of components of another URI in the remote task or one of its superclasses, and all components match; components match if neither is a parameter place-holder and they have the same case-insensitive value, or if either of the components is a parameter place-holder.

HTTP methods

URIs are like other class objects in classes with instances, in that they can have their own methods. There are some special methods supported for URIs, called HTTP methods. These correspond directly to the HTTP protocol methods used by a RESTful API, and they are:

- $delete, $get, $head, $options, $patch, $post, $put

The HTTP methods are named with a leading $ (unlike the HTTP protocol methods) so that they work with the usual Omnis inheritance mechanism. URIs can also have other methods, but these are not HTTP methods and are not part of the public ORA. The name is the only property that determines if a URI method is an HTTP method, so renaming a method can make it become HTTP or non-HTTP accordingly.

HTTP methods of a URI have some special features and properties. The first parameter for all HTTP methods must be named **pHeaders,** and defined as a Field reference. This references a row which contains the HTTP headers received in the RESTful request from the server. The row has a column for each HTTP header. The column names are created by converting the HTTP header name to lower case and removing any - characters e.g. Content-type becomes contenttype as a column name. If more than one header exists with the same name, the headers are combined into a single comma-separated value.

For methods which accept content with the request ($patch, $post, $put) the second parameter must be named **pContent,** which is a Field reference to the content received in the request.

When you create a new HTTP method, Omnis automatically creates the parameters pHeaders and pContent, and it also automatically adds a character parameter for each parameter place-holder in the URI. In addition, you can add further parameters to the method (which must be of type character, Boolean, integer or number). Each further parameter is then expected to be part of the query string in the full URL used to make the RESTful call to the method; if you provide an initial value for the parameter, the parameter is optional in the query string.

When the RESTful call reaches the remote task method, pHeaders, pContent, the place-holder parameters and the query string parameters are all automatically populated by Omnis.

Note that once you have created the method, you can delete parameters which are required at runtime e.g. pHeaders. However, Omnis will detect this and generate an error, either at

the ORA level (see the error log in the browser) or when the client attempts to call the method.

An HTTP method has some additional properties:

- **Nickname**
  A name which must be unique in the set of all HTTP methods for the ORA. The nickname is used to uniquely identify the method in the Swagger definition for the RESTful API. Clients generated from the Swagger definition typically use the nickname as the method name to call in the client interface. When you create a new HTTP method, Omnis automatically assigns a default nickname
- **Input type**
  Methods which accept content with the request ($patch, $post, $put) have a property called input type, where the value is one of:
  empty if no content is to be supplied with the request
  a MIME type e.g. application/xml
  the name of an Omnis schema class in either the same library as the remote task, or another library. A schema rather than a MIME type is identified by the absence of a /. When you use a schema class, the supplied content must be application/json, conforming to the definition in the schema class. Note that the columns in the schema class must be character, Boolean, integer, number, list or row, and when using list or row, the list or row must have a schema class subtype which also conforms to these type rules
- **Output type**
  This specifies the type of content returned by the method when it returns the HTTP status of 200 (OK). One of:
  empty if no content is to be returned
  a MIME type e.g. application/xml
  the name of an Omnis schema class in either the same library as the remote task, or another library. The notes regarding schema classes and the input type also apply to the output type
- **HTTP response codes**
  A list of codes which can be returned by the method. These are the application codes that can be significant to clients; in addition, the Omnis server will return other codes such as internal server error, which should not be specified here. With each code you can specify optional status text and an optional schema class used to specify some JSON that you will return when the method returns this status code

The HTTP method properties affect how Omnis interacts with the HTTP method:

- When calling a method which accepts content with the request, then there are two possibilities:
  The input type is either empty or a MIME type. pContent is a field reference to a binary variable containing the content if any.
  The input type is a schema class. Omnis parses the JSON content and generates a row. In addition, Omnis checks that every column marked as “No nulls” in the input type schema is present in the row. If parsing fails, or the column check fails, Omnis returns

an error to the client. Otherwise, pContent is a field reference to the row generated by parsing the JSON.

- When an HTTP method returns, Omnis treats the return value as follows:
  If the HTTP status code set using $sethttpstatus is 200, then the output type property determines the type of the output.
  If the HTTP status code set using $sethttpstatus is another value, then Omnis looks up the status code in the HTTP response codes, and uses the return type for the status code.
  Omnis uses the output type determined from the HTTP status code as follows:
  If the output type is not empty, then there must be some returned content. Omnis automatically sets the content-type header for the response to either the output type, or application/json if the output type is a schema. In addition, if the output type is a schema, then the return value from the method can either be:
  Binary (not recommended). Omnis looks at the first character of the content, and checks that it is {, as a sanity check to see if it is probably JSON (if the check fails, the client receives an error).
  A row (recommended). Omnis checks that the row is defined from a class with the same name (excluding the library) as the output type (if the check fails, the client receives an error). Omnis then automatically converts the row to JSON.
  If the output type is empty, then there can only be returned content if the method has already added a content-type header using $addhttpresponseheader; otherwise Omnis returns an error to the client.

Simple Types

In a schema class, a list column can now have a so-called simple type as its sub-type. Valid values are <character>, <integer>, <boolean> and <number>. These allow ORAs to define JSON that contains arrays of simple types.

Method Editor

The method editor has additional features for a RESTful remote task. There are new menu items that allow you to:

- Insert a new URI
- Delete a URI
- Insert a new HTTP method
- Rename a URI

These menu items are on the context menu for the method tree, and also in the modify menu in the toolbar, provided that the method tree has the focus.

In addition, when the currently selected method is an HTTP method, the variables panel has two additional tabs: RESTful and RESTful notes:

- RESTful allows you to set the input type, output type and HTTP response codes. The status code grid has a context menu you can use to manage its entries.
- RESTful notes allows you to add documentation notes about the method which Omnis writes to the Swagger definition.

Find and replace works with the new RESTful properties; double clicking on a RESTful entry in the find and replace log will open the method editor with the property selected.

The notation helper now lists column names after you enter the name of a list or row variable with a schema or table class as its subtype, followed by .

Server Properties and URLs

The Server Configuration dialog allows two new properties to be configured:

- **RESTful URL**
  The base URL used to call Omnis RESTful Web Services, e.g. http://www.test.com/scripts/omnisrestisapi.dll Omnis uses this in the Swagger definitions it generates. If empty, Omnis uses http://127.0.0.1:$serverport
- **RESTful connection**
  [POOL,][IPADDR:][PORT]. Controls how the Omnis RESTful Web Server plugin connects to Omnis. POOL is a load sharing process pool name;IPADDR and PORT identify Omnis or load sharing process; if empty, defaults to $serverport

These new properties are stored in the config.json file in the Studio folder of the Omnis tree. These properties affect the URLs stored in the Swagger definitions for ORAs implemented in the server.

If you do not set these properties, then the API will be defined to connect directly to the built-in HTTP server in Omnis.

In order to make a call to an ORA, you need a URI. If you look at an ORA in the IDE browser, you can see how the URIs are constructed by looking at the Swagger definition for a top-level URI path (using one of the hyperlinks immediately below the Resource listing hyperlink). The base path will be something like:

http://localhost:8080/omnisrestservlet/ws/5988/api/phase2/myapi

The initial part of the URL (http://localhost:8080/omnisrestservlet) gets the request as far as the Web Server plugin. The next two components of the URL (/ws/5988) tell the Web Server plugin how to connect to Omnis or the load sharing process. These two components are optional, and can be replaced with the Omnis-server header property described in the Phase 1 documentation; however, if you are likely to be doing cross-domain requests, then it is better to use the /ws/5988 form, since it is guaranteed to be sent with an OPTIONS method request. (Note that the “ws” is a fixed value). The second component (5988) has the general syntax definition:

- nnnn (a port number)
- or ipaddress_nnnn (IP address and port number)
- or serverpool_ipaddress_nnnn

The remaining components are forwarded to the Omnis server: /api/phase2/myapi. The first of these remaining components is a fixed value, which tells the Omnis server that this is a call to an ORA (this first component can also have the fixed value swagger as part of a URL to request a Swagger definition, or it can be of the form LIB.RT (as used in Phase 1 of the RESTful server implementation)). The next two components are the library name and the ORA name.

When connecting directly to the Omnis server, the base URL is something like:

http://localhost:5988/api/phase2/myapi

Finally, when combined with the URI in a remote task in the server, the URL used to call an HTTP method for URI /users/{id} (with no query string parameters) is something like:

http://localhost:8080/omnisrestservlet/ws/5988/api/phase2/myapi/users/1234

ORA Properties and Methods

There are various new notation properties and objects to support ORAs. As described earlier, the remote task has new properties $restful and $restfulapiname. RESTful remote tasks have a $objs group. Specific methods in this group are:

- **$add()**
  $add([cUri]) inserts a URI into a RESTful remote task and returns an item reference to it.cUri must be a valid remote task URI starting with a /
- **$remove()**
  $remove(rItem) delete the URI; rItem is an item reference to the URI to delete

HTTP methods in a RESTful remote task have the following new properties:

- **$httpnickname**
  A simple name for the RESTful remote task method exposed via a URI and HTTP method;it must be unique in the RESTful API;it cannot be empty,must start with an alpha character (a-z or A-Z) and can only contain a-z, A-Z, 0-9 and _
- **$httpinputtype**
  Only applies to RESTful remote task HTTP methods. Empty if no input content is required, or the name of a schema class describing the JSON input object if application/json input is required, or a MIME type if other input content is required
- **$httpoutputtype**
  Only applies to RESTful remote task HTTP methods when they return HTTP OK (200).Either empty if no content is output,or the name of a schema class that describes the output JSON object,or a MIME type for other output content
- **$httpnotes**
  Applies to RESTful remote task HTTP methods only.Notes about the method functionality

In addition, HTTP methods have a new group:

- **$httpresponses**
  Applies to RESTful remote task HTTP methods only.The group of HTTP response objects that define the possible response codes (not including 200 OK) for the HTTP method

To add a new response code to the group, use:

- **$add(iCode[,cText,cType])**
  Adds a new HTTP response code definition for the method and returns an item reference to it

The members of the HTTP response codes group have properties as follows:

- **$httpresponsecode**
  An HTTP response code (100-599, excluding 200)
- **$httpresponsetext**
  Text describing the HTTP response code
- **$httpresponsetype**
  This is the name of a schema class that describes the JSON object to be returned as the result of a RESTful call to the HTTP method which returns the associated response code.Empty means no content is returned

Finally, there are two new properties in $root.$prefs (which are also in config.json):

- **$restfulurl**
  The base URL used to call Omnis RESTful Web Services,e.g.http://www.test.com/scripts/omnisrestisapi.dll. Omnis uses this in the Swagger definitions it generates.If empty,Omnis uses http://127.0.0.1:$serverport
- **$restfulconnection**
  [POOL,][IPADDR:][PORT]. Controls how the Omnis RESTful Web Server plugin connects to Omnis.POOL is a load sharing process pool name;IPADDR and PORT identify Omnis or load sharing process;if empty,defaults to $serverport

Swagger Definitions

Omnis populates the Swagger definitions using the properties of the remote task. The Swagger method summary is the Omnis method description. A schema column with no nulls set to kTrue is marked as a required JSON member in the Swagger model object.

The Swagger resource listing contains various fields that need to be populated e.g. API version number, contact email etc. In order to do this, the Omnis tree contains a default template, and you can also create specific templates for specific ORAs. The default template is the file defaultreslist.json in the folder clientserver/server/restful/swaggertemplates in the Studio tree. You can edit this, or alternatively copy it and create an ORA specific template, which must have the name <restfulapiname>.json, and be stored in a sub-folder of swaggertemplates named with the library name e.g.

clientserver/server/restful/swaggertemplates/lib/myapi.json

Omnis reads the template each time it generates a new resource listing. Omnis keeps the Swagger definitions in step with changes in the environment e.g. when you save a remote task or relevant schema class, or change the RESTful URL or connection property.

You can use swagger-ui (https://github.com/wordnik/swagger-ui) with Tomcat to test your ORAs. Take the dist folder for swagger-ui, drop it into the webapps folder of your Tomcat tree, and rename it swagger-ui. Restart Tomcat. You can then use the URL http://localhost:8080/swagger-ui/index.html#!/path in a browser to open swagger-ui.

If you also place omnisrestservlet in Tomcat webapps (and restart Tomcat), and set Omnis server properties restfulconnection to your server port, and restful URL to http://localhost:8080/omnisrestservlet, you can use swagger-ui without any cross-domain issues.

If you select your ORA in the Web Service Server node of the IDE browser, you can click on the Resource listing hyperlink, and copy the URL from the top of the panel showing the Swagger definition. Paste the URL into swagger-ui and press Explore - you should see your ORA.

Managing Return Values

There may be occasions where RESTful API remote tasks are not able to generate their content as the return value of the HTTP method. For these cases, content generation can be deferred until later, for example, until a threaded worker object completes, or to allow push support, possibly using server sent events and text/event-stream content. In order to do this, there are additional steps. Before returning from the HTTP method (where you would usually return content):

```
Calculate $cinst.$restfulapiwillclose as kFalse
```

This prevents the remote task from closing when you return, and it means that you are responsible for closing the remote task by calling $close() at a later point, or by using the remote task timeout mechanism. Note that it is essential to close the remote task, so that the data connection to the client is closed.

Note that setting $restfulapiwillclose to kFalse will be ignored if an error is detected by the Omnis server as part of request processing.

$restfulapiwillclose has the following definition: If true,the RESTful API remote task will close when the Omnis RESTful HTTP method returns. Defaults to kTrue in a new RESTful API remote task. kFalse only applies when the method executes successfully;you must eventually call $close().

After setting $cinst.$restfulapiwillclose to kFalse, you do not need to return any content, headers or status from the method. If you do return content though, then you also need to set the HTTP status and add any response headers before returning the content. Note that the Omnis server no longer automatically adds the content-length header - this becomes your responsibility if this header is required (in many cases like this it is not).

There is a new remote task method:

- **$sendhttpcontent()**
  $sendhttpcontent(xData) sends the next block of HTTP content (xData) to the client for a RESTful API remote task that did not close

When you are ready to generate the response e.g. in a worker callback, call $sendhttpcontent. The xData parameter differs from content returned from a RESTful HTTP method, in that it is always binary (meaning that you are responsible for generating JSON or encoding characters for example).

You can call $sendhttpcontent more than once, to incrementally send content. However, before the first call, you must set the HTTP status and supply the HTTP response headers (including content-length or transfer-encoding chunked if required).

When using $cinst.$restfulapiwillclose set to kFalse, the Omnis server does not attempt to validate the content returned as it does for JSON content when using $cinst.$restfulapiwillclose set to kTrue.

$sendhttpcontent cannot be used in the initial RESTful API HTTP method call.

Transfer-encoding chunked

You can return content in multiple blocks using transfer-encoding chunked by using $sendhttpcontent. To facilitate this, there is a new built-in function:

- **formatchunk()**
  formatchunk(data) formats the data as a chunk suitable for sending to the client using chunked transfer encoding.Data can be character (which Omnis converts to UTF-8) or binary

Each data block to be sent can be sent with code such as:

```
$ctask.$sendhttpcontent(formatchunk(data))
```

These calls need to be followed by a call to send a zero-length chunk (which terminates) the content:

```
$ctask.$sendhttpcontent(formatchunk())
```

Server Sent Events

You can use $sendhttpcontent to handle a push connection from a client using Server Sent Events. To do this, set the output type for a get method to text/event-stream. Note: you do not need a content-length header for this. You can then send events to the client using $sendhttpcontent. To facilitate this, there is a new built-in function:

- **formatserversentevent()**
  formatserversentevent(fieldname,fielddata[,fieldname,fielddata]...): formats data suitable for sending as an event when generating text/event-stream content.Parameters can be character (which Omnis converts to UTF-8) or binary (UTF-8)

For example:

```
Do $ctask.$sendhttpcontent(formatserversentevent("id",1,"data",”my
   event data”))
```

The protocol field names in the example are data and id, with values 1 and “my event data” respectively.

Web Services Functions

HTTP Headers

There are two new functions to facilitate using date HTTP header values.

- **parsehttpdate()**
  parsehttpdate(httpDate) parses a date value in HTTP header format (e.g. Sun, 06 Nov 1994 08:49:37 GMT) and returns an Omnis date-time value (in UTC) or NULL if the value cannot be parsed successfully.
- **formathttpdate()**
  formathttpdate(omnisDate) formats the Omnis date-time value (assumed to be in UTC) as an HTTP date header value and returns the resulting string.
- **parsehttpauth()**
  *parsehttpauth(*auth*)* parses the HTTP Authorization header value *auth* and returns a row variable containing the extracted information. See *Authorization* section for more details.

BASE64 encoding

There are two new functions for handling BASE64 encoded data. You are recommended to use these with RESTful requests that require them, rather than the functions in OXML.

- **bintobase64()**
  bintobase64(vData) encodes *vData* as BASE64 and returns the result.*vData* can be either binary or character.If *vData* is character,Omnis converts it to UTF-8 before encoding it as BASE64.
- **binfrombase64()**
  binfrombase64(vData) decodes the binary or character *vData* from BASE64 and returns the resulting binary data.Returns NULL if *vData* is not valid BASE64.

Timer Worker Objects

The timer component now contains a Worker Object. This has the advantage over the other timer objects in that it can be used with remote tasks in the multi-threaded server. It has the following properties:

- **$timervalue** and **$useseconds**
  These work as for the current timer objects
- **$repeat**
  If true, then after calling $starttimer() the timer will fire until $stoptimer() is called or the object is deleted;otherwise the timer will fire at most once for each call to $starttimer().A change to $repeat is ignored until the timer is started again

The new timer component supports the methods $starttimer() and $stoptimer(). Just like $repeat, changes to $useseconds or $timervalue do not take effect if the timer is already running.

When the timer fires (or the timer is cancelled), Omnis calls the $completed or $cancelled method in the object, just like other worker objects. This occurs in the context of the task that owns the object, and interrupts any code running for that task (after a complete method command has executed).

Cross Origin Resource Sharing

Cross Origin Resource Sharing (CORS) "is a mechanism that allows many resources (e.g., fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain the resource originated from" (wikipedia). An Omnis RESTful API can handle CORS by implementing the $options HTTP method, and by handling the Origin and other headers when processing other HTTP methods (see above for details). In addition, Omnis Studio 6.1 allows you to configure the Omnis Server (in both the development and server runtime versions) to automatically handle CORS. This means that the Omnis Server can be configured to automatically send the response to OPTIONS, and to add the correct CORS headers to the response buffer before passing a simple or actual request to the application.

CORS configuration can be included as a new top-level section in the config.json file in the studio folder. This can be explained using an example.

```
"CORS": {
    "originLists": {
        "list1": [
            "http://127.0.0.1:8080"
        ],
        "list2": [
            "http://127.0.0.1:8080",
            "http://localhost:8080"
        ],
        "list3": [
            "http://123.456.0.0"
        ]
    },
    "headerLists": {
        "headerList1": [
            "omnis-error"
        ],
        "headerList2": [
            "omnis-error",
            "omnis-server"
        ],
        "chrome": [
            "accept",
            "content-type",
            "omnis-server"
        ]
    },
    "exposedHeaderLists": {
        "exposedHeaderList1": [
            "omnis-error1"
        ],
        "exposedHeaderList2": [
            "omnis-error1",
            "omnis-server1"
        ]
    },
    "APIS": {
        "*": {
            "origins": "list1",
```

```
        "headers": "headerList1",
        "exposedHeaders": "exposedHeaderList1",
        "supportsCredentials": true,
        "maxAge": 0
    },
    "Swagger": {
        "origins": "list2",
        "headers": "headerList2",
        "exposedHeaders": "exposedHeaderList2",
        "supportsCredentials": true
    },
    "PHASE2.myapi": {
        "*": {
            "origins": "*",
            "headers": "headerList1",
            "exposedHeaders": "exposedHeaderList1",
            "supportsCredentials": true
        },
        "Swagger": {
            "origins": "*",
            "headers": "headerList1",
            "exposedHeaders": "exposedHeaderList2",
            "maxAge": 1234
        },
        "/array": {
            "origins": "list3",
            "headers": "chrome",
            "exposedHeaders": "exposedHeaderList2",
            "supportsCredentials": false,
            "maxAge": 1234
        },
        "/second/{uriParameter}/{p2}": {
            "origins": "list2",
            "headers": "*",
            "exposedHeaders": "exposedHeaderList2",
            "supportsCredentials": true,
            "maxAge": 12
        }
    }
```

```
        }
    }
```

In the example, first look at the object members of the CORS object. This can have members as follows (note that everything here is optional, and the most likely result of omitting data is that a request will be passed to the application to handle, or a method not supported error will be returned to the client if the application does not implement the method):

- originLists: Each member of originLists is a named list of origins i.e. possible values for the HTTP Origin header. (Each list is an array)
- headerLists: Each member of headerLists is a named list of HTTP headers (Each list is an array)
- exposedHeaderLists: Each member of exposedHeaderLists is a named list of HTTP headers (Each list is an array)
- APIS: This object has members as follows:
  *: Server wildcard **CORS entry**. See below for the definition of CORS entry
  Swagger: Server Swagger CORS entry
  Entries named library.api. Each library.api object has members as follows:
  *: API wildcard CORS entry
  Swagger: API Swagger CORS entry
  CORS entries named using a URI string. These URI strings need to match URI object names in the API

A CORS entry has members as follows:

- origins: This has either the value * (meaning that when this CORS entry is used, all origins are allowed), or the name of a member of originLists (meaning that when this CORS entry is used, only the origins in the list are allowed).
- headers: This has either the value * (meaning that any header requested by the client using the Access-Control-Request-Headers header is acceptable), or the name of a member of headerLists (meaning that only headers in this list can be requested by the client using the Access-Control-Request-Headers header).
- exposedHeaders: The name of a member of exposedHeaderLists. Headers in this list will be returned using Access-Control-Expose-Headers when handling a simple or actual request.
- supportsCredentials: If true, and the origin is allowed,the server adds Access-Control-Allow-Credentials with value true.
- maxAge: The number of seconds that a client is allowed to cache the result of an OPTIONS method.

CORS processing in the Omnis server occurs when a request with an Origin header arrives. The server tries to locate a CORS entry for the request. There are two cases:

- When the client is requesting Swagger data, the server looks for the API Swagger CORS entry. If the API has no configuration, or no API Swagger CORS entry, the server looks for the Server Swagger CORS entry.

- When the client is executing an API method (resulting in either the method call or an OPTIONS method call), the server looks for the CORS entry exactly matching the URI that will be used to make the request; if that is missing, the server looks for the API wildcard CORS entry; and if the latter is missing, the server looks for the Server wildcard CORS entry.

If the above processing does not locate a CORS entry, then the server does not carry out any CORS processing, and the request continues as it would without CORS. If however the above processing locates a CORS entry:

- The server will attempt to generate the response to OPTIONS, provided that the logic in section 6.2 of the W3C Recommendation referenced earlier applies.
- The server will add CORS headers to the response buffer for other requests, provided that the logic in section 6.1 of the W3C Recommendation referenced earlier applies.

In order to understand what is going on, there is a new log type that you can specify in the datatolog member of the log configuration: "cors". Using this will cause the server to log CORS issues that mean the CORS processing in the server has not handled the request, and is passing it on to the application if possible.

Authentication

You must be responsible for setting up authentication in your Omnis library. When using a real Web Server (rather than the built-in Tomcat server), you can configure the URL for the web service to support basic or digest authentication. There is also the option of using https, and also client certificates to further secure connections.

There is a new function, *parsehttpauth(*auth*)* which parses the HTTP Authorization header value *auth* and returns a row variable containing the extracted information. Column 1 of the returned row (named scheme) is the scheme (e.g. basic). Other columns are scheme dependent. Examples for various auth header values:

- Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
  Returned row has three columns:
  scheme: basic
  username: Aladdin
  password: open sesame
- Digest
  username="Mufasa",realm="testrealm@host.com",nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",uri="/dir/index.html",qop=auth,nc=00000001,cnonce="0a4f113b",response="6629fae49393a05397450978507c4ef1",opaque="5ccc069c403ebaf9f0171e9517f40e41"
  Returned row has 10 columns:
  scheme: digest
  username: Mufasa
  realm: testrealm@host.com
  nonce: dcd98b7102dd2f0e8b11d0f600bfb0c093
  uri: /dir/index.html
  qop: auth

nc: 00000001
cnonce: 0a4f113b
response: 6629fae49393a05397450978507c4ef1
opaque: 5ccc069c403ebaf9f0171e9517f40e41

- OAuth
  realm="Example",oauth_consumer_key="0685bd9184jfhq22",oauth_token="ad180jjd733klru7",oauth_signature_method="HMAC-SHA1",oauth_signature="wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D",oauth_timestamp="137131200",oauth_nonce="4572616e48616d6d65724c61686176",oauth_version="1.0"
  Returned row has 9 columns:
  scheme: oauth
  realm: Example
  oauth_consumer_key: 0685bd9184jfhq22
  oauth_token: ad180jjd733klru7
  oauth_signature_method: HMAC-SHA1
  oauth_signature: wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D
  oauth_timestamp: 137131200
  oauth_nonce: 4572616e48616d6d65724c61686176
  oauth_version: 1.0
- Bearer 0b79bab50daca910b000d4f1a2b675d604257e42
  Returned row has 2 columns:
  scheme: bearer
  token: 0b79bab50daca910b000d4f1a2b675d604257e42
- Any other scheme:
  Returned row has 2 columns:
  scheme: scheme name in lower case
  data: the rest of the header data

Web Server Configuration for Authentication

If you want to setup Basic and/or Digest authentication for your web services running on Tomcat, Apache Web Server or IIS, please refer to the tech notes section on the Omnis website at: www.tigerlogic.com/omnis/technotes

Manipulating Resources

The server can use the same mechanism for manipulating resources as the client, so refer to the sections above. For configuring and logging the Omnis RESTful API server, see the following section which includes information about how you can configure and log the server using the new Omnis configuration file (config.json).

Omnis Server Configuration File

For Omnis Studio 6.1 there is a new JSON based configuration file in the **Studio** folder called **'config.json'.** The new config file is used to configure the Omnis App Server, including setting up properties for the server itself and logging, as well as settings for Web Services support. The new file also includes a section to enable the Java Class cache to be cleared, and other configurable items in Omnis.

The configuration of the Omnis Server can be set up during installation or by selecting the **Server Configuration** option in the File menu in the Omnis Server. Alternatively, you can change the settings by editing config.json using any compatible text editor, but the file must conform to JSON syntax.

Server Configuration

The first part of the config.json file has the following layout:

```
{
   "server": {
      "port": 5988,
      "stacks": 20,
      "timeslice": 1,
      "webServiceURL": "",
      "webServiceConnection": "",
      "webServiceLogging": "full",
      "webServiceLogMaxRecords": 100,
      "webServiceStrictWSDL": true,
      "RESTfulURL": "",
      "RESTfulConnection": "",
      "start": false,
      "retryBind": false,
      "showBindRetryMessage": true,
      "bindAttempts": 10
   }
}
```

where

- **port, stacks, timeslice**
  configure the Omnis Server executable
- **webService**…
  these parameters configure WSDL/SOAP based web services
- **RESTful**…
  these parameters configure REST based web services

- **start**
  if true means Omnis automatically executes Start server at startup
- **retryBind**
  Set retryBind to false if you do not want Omnis to retry binding to the server port after its first attempt; retryBind defaults to true if it is omitted
- **showBindRetryMessage**
  If retryBind is true, showBindRetryMessage controls whether or not a working message is displayed while retrying the bind to the server port
- **bindAttempts**
  If retryBind is true, bindAttempts overrides the default number of attempts to bind to the port at 1 second intervals

Server Logging

Omnis Studio 6.1 also has a new logging mechanism to support the Omnis App Server when it is serving RESTful web services. There is a new external component that performs logging, located in the logcomp folder of the Studio tree. For Studio 6.1, there is just one component, logToFile. You can configure server logging by adding a member to the new config.json file, with the following layout:

```
{
   "server": {
      "//": "See Server Configuration section above",
   },
   "log": {
      "logcomp":"logToFile",
      "datatolog": [
         "restrequestheaders",
         "restrequestcontent",
         "restresponseheaders",
         "restresponsecontent",
         "tracelog",
         "seqnlog",
         "soapfault",
         "soaprequesturi",
         "soaprequest",
         "soapresponse"
      ],
      "overrideWebServicesLog":true,
      "logToFile": {
         "folder":"logs",
         "rollingcount":10
```

```
        }
    }
}
```

where

- **logcomp**
  is the name of the logging component to use, e.g. "logToFile"
- **datatolog**
  is an array that identifies the data to be written to the log - one or more ofthe values listed in the array above
  - tracelog means that data written to the trace log is also written to the new log
  - seqnlog means sequence log entries that record method execution are writtento the new log instead of the old sequence log file
- **overrideWebServicesLog**
  allows you to just send SOAP web service log entries tothe new log. true means just send log entries to the new log, false means send them to both the old web services log and the new log.
- **logToFile**
  is a member with the same name as the value of logcomp. This contains configuration specific to the logging component.
  - folder is the name of the folder where logs will be placed, relative to the Omnis data folder
  - rollingcount is the number of log files that will be maintained. The logcomponent uses a new log file every hour (and a new one at startup). The logcomponent deletes the oldest file or files so that the number of log files does not exceed this count

Each log record has the following layout:

```
{"thread":0,"when":"20141017
14:04:14","type":"tracelog","length":127}ExternalLibrary File
'C:\dev\UnicodeRun\xcomp\damdb2.dll' failed to load. OS Error:
The specified module could not be found.
```

where

- **thread**
  identifies the thread logging the entry,
- **when**
  is the date and time of the entry,
- **type**
  is the type of the entry (one of the datatolog values), and
- **length**
  is the length in bytes of the data following the initial JSON header.

This is followed by a final CRLF. Log files can typically be read in a text editor, but be aware that they can contain binary data if the content of RESTful requests or responses is binary.

Java Class Cache

You can clear the Java Class cache on startup by adding or enabling a property in the new Omnis configuration file. You need to add an entry at the same level as the “log” and “server” entries to enable resetting the Java Class cache:

```
"java": {
   "resetClassCacheOnStartup": true
}
```

The default value of "resetClassCacheOnStartup" is false.

Empty Method Lines

When editing a method and if you click on the grey space at the end of the method, Omnis now adds space for 64 method lines (previously only one line was added). You can edit the method editor configuration in config.json to change this to any value from 1 to 128 inclusive; the following entry must be added at the same level as the “server” entry.

```
"methodEditor": {
   "stripTrailingEmptyCommands": true,
   "blankLinesToAdd": 64
}
```

When the method editor saves a method back to the class, that is, as it is being navigated away from, Omnis strips empty method commands from the end of the method. You can disable this behavior by editing config.json.

Synchronization Server

There are a number of enhancements to the Omnis SQLite Synchronization Server provided with Studio 6.1. The iOS and Android application wrappers provided with Omnis Studio 6.1 must be used with version 2 of the SQLite Synchronization Server. The following features have been added:

Device Recognition

The new Synchronization Server supports automatic device recognition by hardware-id.

User Groups

The Synchronization Server now supports up to 255 user *groups* instead of the previous 255 user/device limit. Devices still connect with a user (group) name and password and the hardware-id is transmitted automatically. Each group supports up to 65535 unique devices and each device is allocated a pool of 65535 primary key values per table where applicable.

Server-Side Replication

The new Synchronization Server supports Server-Side Replication to accommodate external changes to the consolidated database.

64-bit integers and other enhancements

Improved support for 64-bit integers and several enhancements to the Synchronization Server user interface.

Upload or Download Synchronization

The sqlobject.$sync() method now accepts an optional parameter which can be set to 'upload' or 'download' for synchronization in one direction only (works for SQLite only).

Further details

Further details about these enhancements and information about how to use the Omnis SQLite Synchronization Server are available in the 'SQLite Synchronization Server' manual which is available to download from the Omnis website: www.tigerlogic.com/omnis

Transform Component

There is a new 'non-visual' external component, called **Transform,** that allows basic non-blocking animation and transformation effects to be added to standard runtime windows (window classes) and window objects. (Note this is not for JavaScript objects on remote forms).

Transformation can be applied to any numeric window or object property accessible via standard Omnis notation, including background, foreground and external components. Transformation cannot be applied to attributes that require character values (e.g. text and date fields) and cannot sensibly be applied to attributes that use constant values (e.g. colors and Boolean values).

Note: there is a tech note on the Omnis website that provides more information and example libraries for the Transform component:

http://www.tigerlogic.com/tigerlogic/omnis/technotes/tnxm0004.jsp

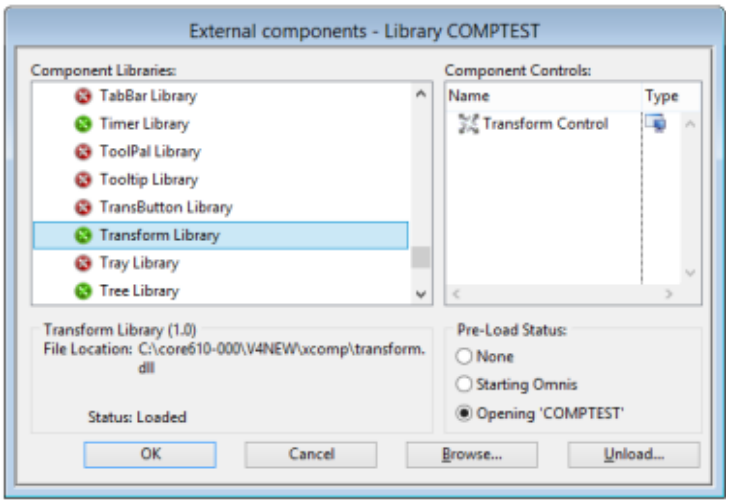
How does it work?

During standard Omnis method execution, assignment of property values is normally actioned immediately. The transform object works by making these assignments incrementally using a specified number of steps. The delay between each step can be specified and it is also possible to specify a 'convergence effect' to be used. Convergence allows a number of additional animation steps to be added that slow or graduate the animation to give the desired visual effect.

There can be multiple transform objects on a single window, each with its own set of 'state' methods and animation settings. Each object uses its own internal timer values, allowing multiple transform objects to execute independently of one another.

Adding a Transform Object

To add a transform object to a Window class in development mode, go to the External Components tab of the Component Store and right-click to open the component library list and select the Transform Library and then the Transform Control.



Once loaded, simply drag the Transform Control (icon) on to the window and rename it as appropriate. If you double-click on the new component, you will note that aside from $construct, $destruct and $event, no default methods are shown. The transform component's primary method; *$transform()* is private and should not be overridden.

Aside from being placed on the window, the transform component has no visual display capability so it may be desirable to set the component's $visible property to kFalse or position it beyond the window's edge.

Creating Transform States

The transform object uses methods to encapsulate each 'target state', that is, after the transform object is placed on the window, one or more methods are created inside the object that represent each state. Each method line takes the form of an assignment statement, e.g.

```
Do $cwind.$objs.object.$attribute.$assign(value)
```

Calculate…as statements may also be used although the transform component will attempt to convert these to Do…$assign() statements. Non-assignment statements, including but not limited to computational, control and conditional statements will be ignored by the transform object and should be avoided.

Assignment statements can include full notation, e.g. $root.$iwindows.myWindow… as well as contextual notation using $cinst and $cwind. Square bracket notation and other context-sensitive addressing may not be used on the *left-hand-side* of the assignment as the

underlying timer object will not have access to context-specific information during execution.

Assignment *values* can include literal values and instance variables including those derived from list and row variables as well as context-specific notation, such as $cwind. This is possible since acquisition of the current and target values (as well as expansion of $cinst/$cwind) is performed during initialization of the transformation.

Wait Statements

When parsing a state method, the transform component additionally recognizes 'wait' statements. These are written in to the method as comments and take the form:

```
; wait  n
```

where n specifies a number of animation frames to wait before proceeding. Wait statements create the effect of staggering the commencement of animation for any statements that follow.

Invoking Transformation

To invoke transformation to a given state, you have to call the object's $transform() method, passing the name of the target method as a parameter, for example:

```
Do $cwind.$objs.myTransform.$transform('$state1') Returns #F
```

Note that it is not sufficient to simply invoke the object's state method directly.

Aside from their use by the $transform() method, there is nothing to prevent transform object methods from being called explicitly by other window objects or via the transform event methods. If you add non-assignment statements to such methods and call them explicitly, these methods will execute as normal. This also allows various transform states to be tested if required.

If $transform is called on an object that is already executing, the current transform terminates and the new transform is calculated based on the current values of the window properties. It is also possible to invoke $transform() from either of the evTransformBegin or evTransformComplete event messages although this should be carried out judiciously in order to avoid unwanted recursion.

Transform Object Methods

The transform object supports a single method that analyzes the specified target 'state' method, builds a list of applicable assignment statements and compiles them into blocks of statements to be executed during each animation frame. The compiled statement list is then passed to an internal timer object which processes the list until transformation is complete.

Method	Description
$transform()	$transform(cState) invokes transformation from the window's current state to the specified state. cState is the name of a private or public ($) method inside the transform object. $transform() returns kTrue on success, kFalse otherwise.
$event()	The transform object supports two custom event messages: evTransformBegin and evTransformComplete. evTransformBegin is called immediately when $transform() is called. evTransformComplete is called when the transform execution list is exhausted. In either event, the method name representing the target state is passed via event parameter 2. If $transform() is called on an object that is already executing, evTransformComplete is not called for the current transform and evTransformBegin will be called for the new target state instead.

Transform Object Properties

Property	Description
$animdelay	The delay in milliseconds between each frame of the transform animation. The default value is 20ms.
$numsteps	The number of steps required to complete the transform (default 20). Note that when non-linear convergence is specified, this is an approximation only since additional steps are automatically added to facilitate the decreasingly smaller step sizes which occur during convergence.
$convergence	Specifies the type of convergence to be used as animation completes. The default value; kConvergeSine causes the step size to decay gradually/sinusoidally. kConvergeLinear turns-off convergence and the step size will be constant. kConvergeOvershoot employs decaying sinusoidal convergence to create an overshoot/bounce effect.

Miscellaneous Enhancements

Oracle DAM

Fetching Very Large Objects using the Oracle DAM

The Oracle DAM now has the ability to fetch very large objects (up to 2GB) directly to the local file system. Two new properties have been added to the session object:

- $filethreshold
  the file threshold which is initially set to 50MB
- $filedirectory
  the directory to receive the file which is initially set from the USERPROFILE environment variable on Windows or HOME on Mac and Linux

Any CLOB, NCLOB, BLOB or BFILE column which exceeds $filethreshold will now be fetched in chunks using $lobchunksize directly to $filedirectory. The filename used will conform to "colname_xxxxxx.BIN" where xxxxxx is a unique identifier (based on #CT). Any character data written to file will be converted to UTF8, otherwise raw data will be written. For BFILEs this means that changing the file extension later (e.g. from .BIN to .AVI) will result in a facsimile of the original file. When a VLOB is written to file, its filename is returned into the result list column. Since the result column was previously described as binary it is necessary to extract the filename using the utf8tochar() function, e.g. Calculate filename as utf8tochar(lResult.1.colLOB).

Using Worker objects to fetch VLOBs

Fetching VLOBs on the main thread can cause Omnis to pause while the data is being transferred. Therefore, for large transfers it may be preferable to SELECT and FETCH each VLOB using an Oracle worker object. The main thread is then free to continue and will be notified when the VLOB has been fetched.

ODBC DAM

The $datesecdp property specifies the number of decimal places used for server date columns. The property is now set to zero for a MyODBC connection to allow correct type mapping to DATETIME.

Debug Session Files

The $debuglevel property specifies the level of debug information written to the debug file specified in $debugfile. The $debuglevel property can now be set to 5 which causes a time stamp to be prepended on to each debug entry. The time stamp is accurate to 1/60th second and reflects the time since the session object logged-on. Debug lines written before $logon() reflect the system up-time.

Session Object Properties

There are two new properties $sessionobjref and $statementobjref of a list or row defined from a SQL table, and one new property of a session in $sessions called $sessionobjref. These are equivalent to $sessionobject and $statementobject, except that they work exclusively with object references.

Object References Auto Delete

Object references are now deleted automatically when they are no longer required in order to free up memory. Object references are deleted when a variable or list column no longer contains the reference. Therefore calls to $deleteref are no longer required unless you want to release memory sooner than would otherwise occur under the automatic process.

Statement Methods

The $columns() statement method has a new parameter to better describe the columns in a server table. The method $columns(cTableName[,iFlags]) generates a result set describing the columns of the specified table – the rows in the results set describe each of the columns in the table, including the column length in the 8th column of the list. In Studio 6.1 an optional flags parameter can be specified to generate column lengths for Number, Integer and Date columns. Values to be ORed together can be found in the Omnis Catalog under the Statement Flags group. Without the iFlags parameter, the default behavior is retained, i.e. lengths will only be returned for Character columns.

Measuring Data Transfer

You can now measure the amount of data (in bytes) that is received and sent through a session object since logon using the new session properties $bytesreceived and $bytessent. The values can be reset by assigning zero to them. These properties apply to all DAMs.

Page Print Preview

There is a new Page print preview window that allows the end user to select text from the screen and review pages in a page list in margin of the preview window.

Max Number of Method Lines

The limit of 1024 lines in an Omnis method has been removed. However, although the number of method lines is theoretically unlimited now, the maximum number of method lines is capped at 256,000 to maintain efficiency in your code.

Commenting Multiple Methods

You can now add the same comment to multiple selected lines of code. To do this you need to copy the initial comment or text, select the required method lines, right-click on the selected methods and select the "Paste To Inline Comment" option in the context menu. The comment or text will be added to the inline comment for each selected method.

Comparing Variables

You can now do comparisons in the Omnis language between *binary* variables, *object* variables and *object reference* variables, when both sides of the operator are the same type.

Binary comparisons compare the data byte by byte until there is a non-matching byte, in which case the first variable is greater than the second variable if the non-matching byte in the first variable is greater than that in the second variable. The comparison extends to the length of the shortest variable; if all bytes match, then the first variable is greater than the second if it is longer than the second, and vice versa.

Object comparisons compare the object instance – if the instance is the same, the variables are equal.

Multi-threaded Language Separators

The main Omnis App Server thread and any other server thread(s) can now have their own values for decimal point, thousand separator, and import dp, which are stored in the $separators Omnis root preference. This allows you to support multiple languages in a single app running on the Omnis App Server.

If you call $separators from an Omnis App Server thread, the new values for the function parameter and import separators are ignored – you can only set these two separators when running in the main thread.

In addition, once you have started the server with the *Start server* command, subsequently changing the language only affects the decimal point, thousands separator and import decimal place for the main thread.

Web Services Strict Mode

The following relates to the existing WSDL based Web Services component, and not the new RESTful implementation.

When importing a WSDL using the Class Wizard in Omnis, there is now a new **Strict Mode** option. If enabled (the default), the WS object will be generated in the same way as previous versions. If you uncheck this, any "wrapped" simple types in the WSDL will be treated as their base types.

This means that where before a simple character type defined in the WSDL, which was wrapped with some restrictions (e.g. max length/min length), would have been imported into Omnis as a character column inside a row variable it will instead be imported into Omnis as a straight character variable.

This enhancement will make things simpler when creating clients, but must be weighed up against whether the loss of restriction information when creating the client is an issue, although only a small subset of restriction information (if any) is translated into the Omnis datatypes.

Popup Menus

The *Popup menu* command has some new parameters that allow you to specify the position of the popup menu. When you run this command you can pass the x and y coordinates of the top-left corner of the menu. If you omit the x and y coordinate parameters (or they are #NULL) the menu will open in the current mouse position.

Strip Spaces in Entry Fields

Single line entry fields and Combo boxes for window classes have a new property, called $stripspaces, which when true strips leading and trailing spaces from the data before storing it in the variable or field. This property is set to kTrue to maintain compatibility with previous versions, which means leading and trailing spaces are stripped from data. If however you want to retain the exact data that is entered by the user, including any leading and trailing spaces, you need to set this property to kFalse.

HTTPPage

The HTTPPage command has an additional parameter to allow you to ignore SSL. The full syntax of the command is:

```
HTTPPage (url[,Service|Port,pVerify]) Returns html-text
```

When passed as false, the pVerify argument prevents SSL verification when using a secure URL, so you can use:

```
HTTPPage (url,,kFalse)
```

Transbutton Hot Tracking

There is a new property, $nodrawhotrect, in the Transbutton external component (window class component, not JavaScript) to prevent the rectangle drawing during hot tracking – the property is set false by default for compatibility, which means the rectangle is displayed. Note that the hot rectangle is not displayed on OSX, so this property has no affect when running on OSX.

Library Startup Task

The Omnis root preference $clibstartuptask has been added which reports the startup task for the library containing the current executing method.

sleep() Function

A sleep() function has been added to allow you to suspend method execution for a specified length of time. The function sleep(milliseconds) suspends execution for the specified number of milliseconds, and returns true if execution was suspended successfully or false if an error occurred. Note this function is not available in client-executed methods.

Printing Sections

The $printsection method has a new parameter to force the report section to print as a record section. The method $printsection(iSection[,bPosnIsRecord=kFalse]) prints a section: note bPosnIsRecord applies to positioning sections only. If bPosnIsRecord is kFalse (the default), this method prints a section based on the position of the previous section; otherwise, when this parameter is true the method prints the section as a record.

Find and Replace Log

There is a new optional 6th argument (bClearLog) for the $findandreplace method, which when kTrue clears the log first before adding the next entry. The new bClearLog flag defaults to kFalse which means entries will be appended to the current log contents.

MailSplit

If the encoding cannot be determined from the MIME, MailSplit now uses $importencoding as the default encoding rather than UTF-8, provided that $importencoding is an 8 bit encoding, that is, anything except kUniTypeUTF16, kUniTypeUTF16BE and kUniTypeUTF16LE.

What's New in Omnis Studio 6.0.1

Omnis Studio 6.0.1 provided several enhancements in the JavaScript Client technology to make the creation of web and mobile applications easier and quicker. The Omnis Studio 6.0.1 release included the following features and enhancements:

- **More Screen sizes and Devices supported**
  Support for multiple screen sizes and devices in the JavaScript Client has been extended to include the BlackBerry® Q10 and the Samsung® Galaxy S4, plus developers are now able to request new sizes
- **Local Database support and Synchronization for Android**
  local database support has been added to the Android application wrapper to allow standalone (serverless client) apps with the ability for data and application content synchronization
- **New Trans button control**
  new JavaScript control that can display a different icon and back color when the end user's mouse hovers over the control, or when the button is tapped on touch devices
- **Improved performance for String Tables**
  String tables are now converted to separate JavaScript files which are passed to the client; this improves performance for large string tables in multi-language apps
- **Testing different mobile layouts in Firefox**
  Remote forms can be tested with 'Responsive Web Design' mode in Firefox which allows you to try different remote form layouts in a desktop browser
- **New date and other functions for JavaScript client methods**
  New functions to replace hash variables which can be executed in client methods, which allows you to return or set the value of short dates and times, among other things
- **Multiple formats for $dateformatcustom**
  now you can specify multiple date formats in an entry field in a remote form, providing more flexible data entry
- **Shorthand method to add columns to a list**
  new list/row method $addcols() which provides a short-hand way of adding columns to a list or row variable
- **Miscellaneous Enhancements**
  including $extraspace for JavaScript Tree controls, $dataname for JavaScript Labels, the dadd() function now works in client methods, icons in PDF report text, inherited object methods, $backiconid for reports, and TLS support for SMTPSend and POP3 commands

Screen Sizes and Devices

JavaScript remote forms are able to support multiple tablet and phone screen sizes which are stored in a single remote form design layout. Support for different screen sizes has been extended to cater for a number of new mobile devices with different screen sizes or form factors, including the BlackBerry® Q10 and the Samsung® Galaxy S4. The screens on such devices have a specific physical size, but due to their high resolution they may have different CSS pixel dimensions which are used when designing the layout of remote forms. For example, the Samsung Galaxy S4 has a 5" HD display which is 1080 x 1920 physical pixels at 441 ppi resolution, equating to CSS pixel dimensions of 360x640.

Typically you would design the layout of a single remote form for a number of different devices or screen sizes, and then use the floating edge, scaling and centering properties to position and size the form on devices which have sizes relatively close to those that are stored in the form. However, to cater for the new BlackBerry and Samsung devices, the remote form property $screensize has been enhanced with the addition of the kSSZjs345x345 and kSSZjs360x640 (Portrait/Landscape) constants. Plus we have have made it easier to implement new screen sizes between major releases so developers can request new remote form sizes to be added.

Enabling new screen sizes

Thers is a new library preference, $designedscreensizes, which is a comma-separated list of screen size constants for JavaScript Client based remote forms in the library (note this is a library wide preference, not a remote form property). To enab le the new screen sizes, you need to check the kSSZjs345x345 and kSSZjs360x640Portrait options under the Prefs tab in the Property Manager for the current library: note that by enabling the 'Portrait' option you enable the equivalent landscape layout for any given size. The screen sizes enabled in the new library preference will be used to populate the $screensize property in the Property Manager and will be available for all remote forms in the library.

If you change $designedscreensizes, the size and position information for sizes or layouts no longer in the list of designed sizes will not be lost: the designed layouts will remain stored with the remote form in the library.

Implementing new screen sizes

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

If you change the screen sizes supported in the $designedscreensizes library preference, all the HTML files for all remote forms in your library need to be rebuilt to reflect the new set of screen sizes: this is done automatically when you test a remote form since the HTML file is rebuilt every time you test a remote form.

Note there is a new jsctempl.htm template file to accommodate the new screen sizes and ongoing support for any new sizes.

Requesting new screen sizes

Developers can now request a new screen size or form factor for inclusion in Omnis Studio, although we will reserve the right to decide whether or not any suggested screen size is appropriate for Omnis. Between major releases, we will make new sizes available by supplying new screen size configuration files on request. An updated 'ssz.cfg' file will be provided and can be placed in the 'studio' folder in the Omnis tree, and a new 'ssz.js' file which should be placed in the 'scripts' folder. Then for the next major release the new or updated screen sizes will become available for all developers with an updated set of files.

Local Database for Android

Local database support has been added to the Android application wrapper to allow standalone (serverless client) apps with the ability for data and application content synchronization. See the 'Standalone Mobile Apps and Synchronization' section in the 'What's New for Studio 6.0' chapter in this manual for further details about using a local database and synchronization.

Trans Button Control

The **Trans button** is a new JavaScript control that can display a different icon and/or background color when the end user's mouse hovers over the control, or when the button is tapped on touch devices. In all other respects the Trans button is like a standard push button control, insofar as it generates a single evClick event when the button is clicked which can be used to initiate an action in your code. Note the evClick event must be enabled in the $events property for the control for it to be reported.

The Trans button has several properties prefixed "$hot" that relate to the appearance of the button for the hover action. You can specify two icons for the Trans button: one to represent the "off" state which is specified in $iconid, and the other to represent the "over" state which is specified in $hoticonid – if no $hoticonid is specified the $iconid is used. You can also specify a different background color for the hover action in $hotbackcolor, and an alternative border color in $hotbordercolor.

Localization

String Tables

Some developers have experienced performance issues when using large string tables in Omnis Studio 6.0, so in this version string tables are converted to separate JavaScript files and transferred to the client as needed and depending on its locale. The script file is cached in the client browser and only reloaded when the string table has changed.

When you test a remote form that has an associated string table (specified in $stringtable in the remote task linked to the remote form), Omnis generates a JavaScript file automatically if the file does not exist or if the string table file (.tsv) is more up to date than the script file. In addition, Omnis inserts a script tag for the string table into the HTML file generated automatically by the Test Form option. The path of the string table JavaScript file is of the form:

```
html/strings/libname/file.js
```

where 'strings' is a new folder in the html folder, 'libname' is a folder for the library, and file.js is the JavaScript file for the remote task strings, named using the name of the task string table file.

For deployment, you need to place the file.js in the equivalent folder in the web server tree where the other Omnis HTML pages, scripts, etc are located. Alternatively, you can use an option in the String Table Editor to export the string table JavaScript file, rather than using the exported file from your development tree. This option can be used to output the entire table as a JavaScript file, or you can output one or more files for selected locales, where each file contains a single selected locale column and is named file.locale.js.

Optimizing string tables

Single-locale JavaScript string tables can be used to further improve loading performance for string tables. There is a new file, jsStringTableSwitch.htm in the html folder in the main Omnis development tree. This file can be used as the initial remote form for an application, and has markers where it can be customized - this allows you to specify the string table file to use for each locale, and a default for unknown locales. In addition, jsStringTableTempl.htm needs to be customized to set up the initial remote form etc, and the string table path. When a page based on jsStringTableSwitch.htm loads, the page:

- Runs a script which selects the string table file to use, based on the locale.
- Loads the template based on jsStringTableTempl.htm using AJAX.
- Sets the string table to use in the template
- Replaces the document content with the modified template

This results in an HTML page for the remote form that only loads the strings for the current locale, and which still has the original URL you have chosen for your application.

Testing Mobile Layouts

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

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

Date Functions

Omnis has a set of built-in global variables called *hash variables* (since their names begin with #) including a set of date and time variables and other state constants. In this version there are a number of new functions to provide you with access to the values stored in the hash variables from within client executed methods in the JavaScript Client. Note that the default value of these variables may depend on the language version of Omnis Studio you are using.

The date and time related functions use the special date format characters listed under the 'Date codes' item on the Constants tab in the Catalog (press F9/Cmnd-9). For further information about the hash variables, refer to the *Omnis Notation Reference* manual under the 'Omnis Root' section.

- **fmtshortdate()**
  fmtshortdate([*newformat*]) either sets #FD to *newformat* and returns the previous value of #FD, a string used to format Short dates; or if no parameter is supplied returns the current value of #FD (default value is 'D m y')
- **fmtshorttime()**
  fmtshorttime([*newformat*]) either sets #FT to *newformat* and returns the previous value of #FT, a string used to format Short times; or if no parameter is supplied returns the current value of #FT (default value is 'H:N')
- **fmtdatetime()**
  fmtdatetime([*newformat*]) either sets #FDT to *newformat* and returns the previous

value of #FDT, a string used to format Long date and time values; or if no parameter is supplied returns the current value of #FDT (default value is 'D m y H:N:S')

- **fmtdp()** (not available in client executed methods)
  fmtdp([*newdps*]) either sets #FDP to *newdps* and returns the previous value of #FDP, a numeric variable which specifies the format used for display or string conversion of a floating point number; or if no parameter is supplied returns the current value of #FDP (defaults to 12)
- **getdatetime()**
  returns the current system date and time, formatted using #FDT
- **getticks()**
  returns the number of ticks elapsed since system boot; a tick is 1/60th of a second. The value can overflow and restart at zero (for a JavaScript client executed method, the number of ticks since midnight on 1 Nov 2011).

The JavaScript client uses #FD and #FT for short date and short time instance variables in client methods; previously it used #FDT when converting short dates and times to character data.

Error functions

The following functions provide access to the error reporting variables:

- **errcode()**
  returns #ERRCODE, a numeric variable containing the error number generated by a method
- **errtext()**
  returns #ERRTEXT, a string variable containing the error text generated by a method

Key press functions

The following functions report on end user key presses:

- **ctrl()**
  returns true if the control key is being pressed
- **shift()**
  returns true if the shift key is being pressed
- **alt()**
  returns true if the alt or option key is being pressed
- **cmd()**
  returns true if the cmd key is being pressed

Other functions

There are a number of functions that provide access to specific hash variables in your client executed methods:

- **useradians()**
  useradians([useradians]) either sets #RAD and returns the previous value, or if no parameter is supplied returns the current value of #RAD (the default is degrees. If you set #RAD to true, angles are in radians)
- **flag()**
  returns the status of #F, the Omnis flag, which can be true or false

Custom Date Formats

You can now specify multiple date formats in the $dateformatcustom property for entry fields in a remote form, which allows end users to input a date using one of a number of possible formats, rather than being limited to a single date format. The multiple date formats can be entered into $dateformatcustom separated using "|" (the pipe character), for example:

```
D/M/y|D m y|d-m-y|D/M/Y|D/m/y|D-M-y|D M y
```

When parsing data entered by the user, the client uses each format in order, until one successfully matches the user input. The client uses the first format in the list to format the data for display.

Lists

Adding columns

There is a new list/row method $addcols() which provides a short-hand way of adding one or more columns to a list or row variable. It has the following parameters:

```
list.$addcols(cName,type,subtype,maxlen,...)
```

which can be used to add one or more columns to a list or row variable, so the parameter count must always be a multiple of four. Each new column must be specified with the following four parameters:

- **cName**
  the name of the new column
- **data type**
  the Omnis data type represented by one of the type constants, such as kCharacter; all data types are allowed except the Object data type (kObject), since lists of objects are not recommended (you should use object references)
- **subtype**
  the subtype of the new column; only applies to some major types

- **maxlen**
  for some major types such as Character you can specify the maximum length

Miscellaneous Enhancements

JavaScript Tree control

The $extraspace property has been added to the JavaScript tree control to allow you to add extra space in between the lines in the list. $extraspace is the number of pixels added to the normal font height of a row in the list, or zero for no extra space.

JavaScript Labels

The $dataname property has been added to the JavaScript label object which means its text can be populated dynamically. If $dataname is set, its value overrides the value in the $text property, and actually sets $text, so if you read $text when using a label with a data name, the value of $text will become the content of the variable in $dataname.

dadd() function

The dadd() function can now be used in JavaScript client executed methods. The dadd(*datepart*,*number*,*date*) function adds a *number* of "date parts" to the given *date*. The *datepart* parameter is a constant, one of the following: kYear, kMonth, kWeek, kQuarter, kDay, kHour, kMinute, kSecond, kCentiSecond. See the *Omnis Function Reference* for further details.

Icons in PDF report text

You can include icons in text in a report printed to PDF using the style() function and the kEscBmp escape constant. For example, you can use con(style(kEscBmp,1400),'some text') in a report entry field calculation to display an icon on a report.

Icons folder name

You can add your own HD icons to Omnis my creating an icon set and adding it to the icon folder within the html folder in the Omnis tree. However, the icons folder name 'html/_icons' caused some issues with certain versions of Android so has been renamed to html/icons (the underscore has been removed).

See the 'Component Icons' section under 'What's New in Omnis Studio 6.0' for more details about creating and implementing your own HD icons for JavaScript components.

Inherited Object Notation

You can now access the methods of inherited objects in remote forms, windows, menus, toolbars and reports in your Omnis code. Inherited objects are exposed as a new group within the notation group called $inheritedobjs. The members of this group are the inherited objects from all of the superclasses of the class.

Each member of the $inheritedobjs group has three properties: $name, $ident and $isorphan (true when the object no longer belongs to a superclass, but has methods so it cannot be removed without developer approval). In addition, each member has a child $methods group which are the methods implemented in the class for the inherited object. This is just like any other methods group, and methods can be manipulated as you would expect. To override a method from the superclass, simply add a method with the same name as the inherited object method. To inherit a method, delete the method with the same name from the inherited object methods.

Report back pictures

The $backiconid property has been added to report classes to allow you to assign an image to the background of a report using an image ID. The property must refer to an icon ID in an icon set, or an alpha page in an icon data file/#ICONS. $backiconid takes precedence over $backpicture.

TLS support for SMTPSend and POP3

The SMTPSend and POP3 commands now support Transport Layer Security (TLS), a secure network protocol, including STARTTLS for the SMTPSend command, and STLS for the POP3 commands. STARTTLS establishes an SSL connection after the initial socket connection to the SMTP or POP3 server has been established. You can request this when issuing a command that connects to the server, using the optional Secure parameter, which has the possible values:

0 means not secure

1 means immediately secure

2 means connect and then use STARTTLS/STLS to make the connection secure

OEM character conversion

The functions sys(228) and sys(229) have been added to enable and disable tab to tab conversion for OEM character conversions.

SQL workers

The SQL Worker objects now use bind variable names to reference columns in the list/row supplied to the $init() method. See later in this manual or the *Omnis Programming* manual for more information about the SQL workers.

What's New in Omnis Studio 6.0

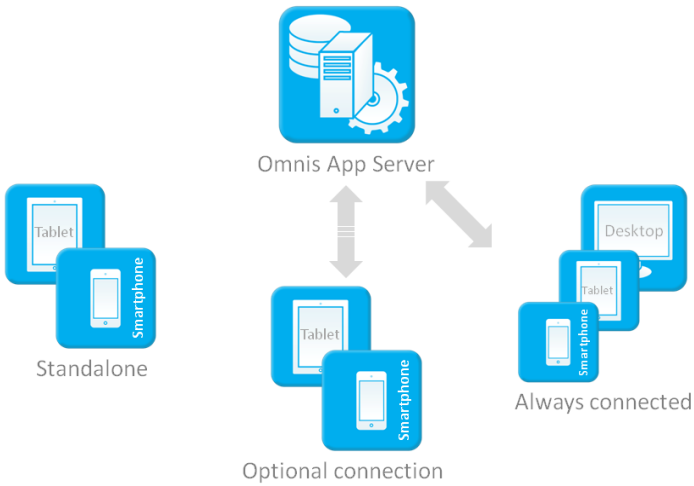
Omnis Studio 6.0 builds on the *JavaScript Client* technology introduced in Studio 5.2 and provides a richer, more interactive experience for your end users on virtually any device, or on any platform. This latest release enhances the capabilities of JavaScript based Remote Forms and ready-made JavaScript Components, and includes the following new features:

- **Standalone Mobile Apps and Synchronization**
  There is a new Wrapper application for the JavaScript Client, available for Android, iOS, and BlackBerry, that allows you to provide mobile applications that can run either without any connection at all to the Omnis App Server, or with an intermittent connection, which would then allow for end user data and application content to be synchronized with the server backend
- **Accessing Mobile Device Features**
  There is a new JavaScript control, called the **Device Control,** that allows you to access hardware features and services on the end user's mobile device, such as the Camera, GPS, Email, Texting, and Contacts info (the new Device control only works in apps running inside the application wrapper)
- **SQL Multi-tasking and SQL Workers**
  The new SQL Worker Objects in the DAMs will allow you to optimize server management by running long SELECT statements or other tasks in the background and on multiple threads, allowing the Omnis GUI to continue without interruption
- **Resizable Remote Forms and Components**
  JavaScript remote forms now have the $resizemode property which controls how a form can be resized in the end user's desktop browser. Plus a new $dragborder property allows JavaScript components to be resized dynamically when the end user resizes the browser window
- **Subform Sets**
  You can now create a special kind of JavaScript based subform, or set of subforms, each with a title bar and resizable borders, that can be opened in the "main" JavaScript remote form; the subforms can be moved, resized, and minimized by the end user
- **High Definition Component Icons**
  You can now use high resolution images (icons) for the controls in your JavaScript remote forms, suitable for displaying on the latest high definition smartphones and tablets; the images are stored in the Omnis tree as separate image files and referenced using the existing Icon ID method

- **Localization Enhancements**
  You can now store localization String Tables as Tab Separated Value (TSV) formatted files alongside your library on the Omnis App Server
- **PDF Printing**
  A new printing device allows you to print a report to a PDF file which can be displayed in your JavaScript web or mobile apps
- **Rich Text Editor**
  There is a new JavaScript control that allows the content in a field to be edited by the end user; word processor like tools are provided for editing the text
- **Dynamic Tree Lists**
  The content inside a Tree List Control can now be built dynamically as the end user expands a node, rather than initially having to build the whole contents of the tree.
- **Linked Lists**
  changes to the Edit and List controls allow you to create a new type of dynamic list that updates itself in response to what the end user types into the associated edit box
- **Omnis VCS**
  a number of enhancements have been made to the Omnis VCS including the addition of Project folders and new preferences for showing only Checked Out classes
- **Numbers and Date Formatting**
  Formatting for Number and Integer JavaScript form fields has been introduced, and formatting for Time and Date fields has been modified
- **Custom CSS styles**
  create your own CSS styles and apply them to the objects in your web and mobile apps, allowing more control of styling and overall design of your apps
- **Miscellaneous enhancements**
  Including 64-bit integer number data types, formatting for data grid columns, $init() and $term() client-side methods for remote forms, and fully justified report text

Standalone Mobile Apps and Synchronization

Omnis Studio 6.0 contains a new wrapper application for the JavaScript Client that allows you to provide mobile applications that can run either "offline", without any connection to the Omnis App Server, or in "online" mode which would allow end users to temporarily connect to the Omnis App Server to synchronize their data and application content.



- **Standalone** – the new wrapper app allows your mobile apps to run completely standalone or "offline" without ever connecting to the Omnis App Server or a database server. The Application Files (remote form definitions, scripts, etc) are bundled with the wrapper app to create a single, clickable application file; this allows complete "offline" operation, or if they are not included in the app bundle a one-off connection can be made to install the app files and from there on a connection to the Omnis App Server would not be needed
- **Optional connection** – the new wrapper app allows your apps to run "offline", but the end user has the option to switch to "online" to synchronize the database and app content via the Omnis App Server; this mode would suit end users who have intermittent connection but need to synchronize their data with a central location
- **Always connected** – apps can be run in the JavaScript Client without the wrapper app via the web browser on the device; in this mode apps must always be connected to the Omnis App Server (the same as current version); apps can run inside a wrapper and remain connected or "online" at all times

Whether you decide to create an entirely standalone mobile app or one that requires synchronization will depend on your intended market and the expectations of your end

users, but with the new JavaScript Client wrapper you now have many possibilities and combinations for creating all types of mobile applications for different devices and business scenarios.

The JavaScript Serverless Client

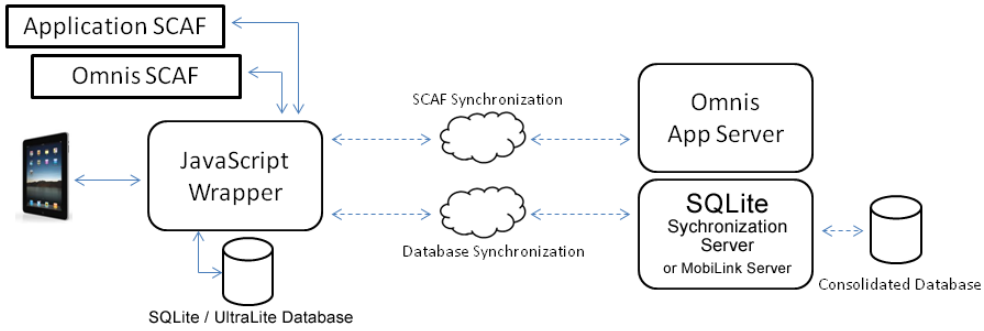
The standalone or offline mode of the JavaScript Client Wrapper application has been dubbed the "Serverless Client" since it can operate entirely without the Omnis App Server. The Serverless Client is a completely self-contained wrapper application for mobile devices which will run Omnis remote forms with no need for any connection to the internet or the Omnis App Server. These apps will have the ability to switch between "offline" and "online" modes at the user's request. Separate forms can be used for the same app in offline/online modes to provide different functionality.

The Serverless Client also includes local database support, utilising an Ultralite database (from Sybase), and even provides the ability for your offline form to synchronize your local database with an online database. This means that a user could switch to offline mode while in areas of no or patchy network coverage to continue working, and then switch back to online mode when back in the office. It is also possible to create completely standalone apps which do not rely on the Omnis App Server and contain all the necessary forms etc in the wrapper application bundle.

PLEASE NOTE: you will require a new alternative Development license to enable the $serverlessclient property in a remote form: the property will remain grayed out without this alternative license.

How does it work?

The Serverless Client (SC) has provision for a local client-side database. This can be used as a local database for the standalone mode for your app, or it can be used to synchronize with an online "Consolidated Database" (CDB). In the latter case, the local database is used for storing tables held in the server-side database and for caching SQL transactions performed whilst the Omnis App Server is not available.



JavaScript Serverless Client allows forms to work in 'offline' mode

The client-side database can be SQLite and the synchronization of the local database with a "consolidated database" administered by the SQLite Synchronization Server provided by

TigerLogic. Alternatively, you can use an UltraLite database from Sybase on the client and a MobiLink server to synchronize the data on the backend.

You can download the 'SQLite Synchronization Server' manual from the Omnis website: www.tigerlogic.com/omnis/download

Standalone Remote Forms

In order for a remote form to be available in standalone or offline mode, you must set its **$serverlessclient** property to kTrue. All methods within the form must be set to 'Execute on Web Client' (right click the method name in the method editor and select 'Execute on Web Client').

Initialization and Termination Methods

The $construct() and $destruct() methods in a remote form cannot be executed as client methods, therefore you can create methods with the names $init() and $term() which perform a similar function that can be executed on the client. The $init() and $term() methods can be used in standalone apps running inside the JavaScript Client wrapper application in which all methods must be executed on the client.

The $init() method is called after the form and the client script files have been loaded. This allows you to do any final initialization of the remote form. The $term() method is called when a remote form instance destructs.

You can use $cinst.$screensize in client methods, including $init(), to get the current screensize of the client.

Serverless Client Application Files

The Serverless Client Application File (SCAF) is a SQLite database that contains all of the resources necessary for a mobile application to run locally in the wrapper in standalone mode. These resources include JavaScript scripts, CSS files, image files and Omnis remote forms. There are two SCAF files needed for each wrapper application:

- **Omnis SCAF** (omnis.db)
  files needed to run the JavaScript Client
- **Application SCAF** (<library_name>.db)
  contains all your application files

Omnis Studio will generate these SCAF files automatically in the '/html/sc' folder under the main Omnis folder. These files need to be placed on the Omnis App Server in the same location for end users to access if necessary (see below).

Whenever you save a remote form which has $serverlessclient set to kTrue, it will update the application files in your Omnis folder. A message is displayed while Omnis exports all the necessary files.

Deployment of SCAF Files

When the wrapper application is executed for the first time in offline mode it will check if any SCAF files are bundled with the application (see the wrapper building document for info on how to do this). If these files exist, they are copied into the application space on the

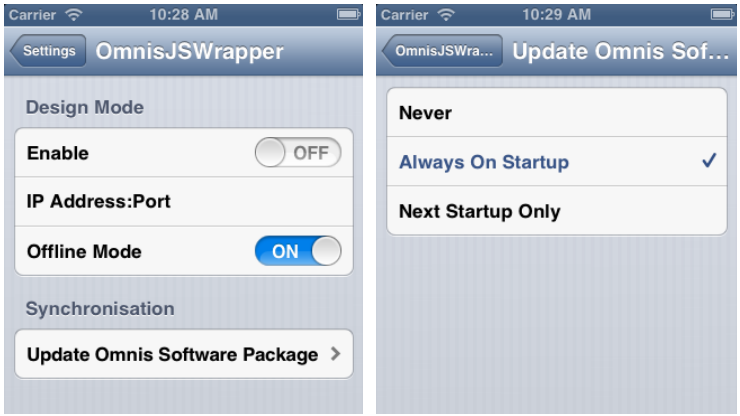
device and used in the wrapper. Note that bundling SCAF files with the application will increase the size of the application bundle.

If SCAF files are not distributed with the application the application will attempt to connect to the Omnis App Server on startup and download the latest versions of the SCAF files.

For the application to know which application SCAF to use the option:
<APPSCAF>name</APPSCAF> (where name is the name of the Omnis library)
is used in the config.xml file within the wrapper application.

Updating SCAF Files

Once the app has been installed onto a device, it will add an entry to the 'Settings' app of the device, with the same name as your app.



The wrapper configuration sub menu "Update Omnis Software Package" contains options for updating the SCAF files. The available options are:

- **Never**
  the SCAF will never be updated
- **Always On Startup**
  will attempt to connect to the Omnis App Server and update the SCAF files every time the application is run
- **Next Startup Only**
  will attempt this update only on the next execution of the wrapper application

It is also possible for the end-user to update the SCAF from inside the app by swiping down the screen to open the runtime menu and selecting the appropriate menu item.

Database Support

The JavaScript wrapper application contains embedded support for SQLite or UltraLite from Sybase and provides client-executed methods with access to a local private SQL database. The database can currently only be used by Serverless Client applications in offline mode.

All interactions between the JavaScript Client and the wrapper will be asynchronous, so the database API also takes this into account.

Schema and Query Classes

The JavaScript Client-executed method code generator restricts the $definefromsqlclass() method for use with either a query or a schema class name as the first argument although it is still possible to pass a subset of column names required using parameter two onwards. This will allow for example:

```
Do listOrRow.$definefromsqlclass('SchemaName')
```

and the code generator will expand this into JavaScript to define the list or row with the columns from the schema.

Data Types

When creating rows to be used for bind variables, it is important that the data types of the columns in the row match those in the database.

Client methods only provide a 'var' data type when creating variables, which will generally be interpreted as Character type. As such, it is safest to manually add columns to your row, using the function:

```
Do lRow.$cols.$add(<name>,<data type>,<data subtype>,[<length>])
```

e.g.:

```
Do lRow.$cols.$add('Age',kInteger,kShortint)
```

The SQL Object

A Serverless Client application gains access to the embedded database using a SQL Object:

`$cinst.$sqlobject`

For example:

```
Calculate oVar as $cinst.$sqlobject
```

All requests to the SQL object are asynchronous (except $getlasterrortext and $getlasterrorcode), and call a client-executed completion method ($sqldone) in the current remote form instance upon completion. Each request returns an identifier when called and the same identifier is passed as a parameter to the completion method, allowing the request to be identified. Thus, multiple requests may be in progress "simultaneously", although they will only execute serially in the wrapper.

Note that you do not need to provide a $sqldone() method although if you do not, errors may be ignored. On success, the returned unique identifier is positive. A negative value indicates an error code.

In the following sections, oSQL is a Var containing the SQL object returned by $cinst.$sqlobject.

$getlasterrortext()

```
Do oSQL.$getlasterrortext() Returns lErrText
```

Returns the error text of the last operation. "OK" implies success.

$getlasterrorcode()

```
Do oSQL.$getlasterrorcode() Returns lErrCode
```

Returns the error code of the last operation. 0 implies success.

$selectfetch()

```
Do oSQL.$selectfetch(cSQL, lBindVars, iFetchCap) Returns id
```

Executes a statement with a result set (typically select or select distinct) and fetches the initial set of rows.

- cSQL
  is the statement. This may be hand-coded, or the result of $select/$selectdistinct for a schema or query class. cSQL can contain bind variable place-holders in the form @[column_name], where column_name is the name of a column in lBindVars.
- lBindVars
  is a row variable referenced by one or more bind variable markers in the SQL text.
- iFetchCap
  is the number of rows to initially fetch (this can be kFetchAll to fetch all rows in the result set).

For all object methods, note that lBindVars may contain columns not referenced by the SQL text. Only those columns referred to by name in the bind place holders will be read. On completion, $sqldone() is called with the following parameters:

- The request id (as returned by $selectfetch)
- A list containing zero or more rows from the initial result set.

At this point it is your responsibility to copy or populate the appropriate form controls to display the data.

Example1:

```
Do iList.$definefromsqlclass('myQuery')
Do oSQL.$selectfetch($clib.$queries.myQuery.$select, iList, 100)
   Returns id
```

Example2:

```
Do oSQL.$selectfetch('select * from Table1 where age=@[age]',
   lBindVars,100) Returns id
```

$fetch()

```
Do oSQL.$fetch(selectfetchid, iFetchCap) Returns id
```

Fetches more rows from the result set generated by a $selectfetch()

- selectfetchid is id returned by $selectfetch().
- iFetchCap is the number of rows to fetch.

In this case, the returned id will be the same. On completion, $sqldone() is called with the following parameters:

- The request id (as returned by $selectfetch)
- A list containing zero or more further rows from the result set.

$insert()

```
Do oSQL.$insert(cSQL, listorrow) Returns id
```

Inserts one or more rows into a database table.

- cSQL is the insert statement. This may be hand-coded, or the result of $insert for a schema class. cSQL can contain bind variable place-holders in the form @[$column_name], where column_name is the name of a column in listorrow.
- listorrow is the list or row containing the data to insert.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $insert()).

Example:

```
Do oSql.$insert("INSERT INTO Product (name, quantity) VALUES
   (@[colName],@[colQuant])",lBindVars) Returns IDinsert
```

$delete()

```
Do oSQL.$delete(cSQL, row) Returns id
```

Deletes zero or more rows from a database table.

- cSQL is the delete statement. This may be hand-coded, or the result of $delete() for a schema class. cSQL can contain bind variable place-holders in the form @[column_name], where column_name is the name of a column in row.
- row contains the values referenced by the bind variable place-holders.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $delete()).

$update()

```
Do oSQL.$update(cSQL, newRow, oldRow) Returns id
```

Updates zero or more rows of a database table.

- cSQL is the update statement. This may be hand-coded, or the result of $update for a schema class. cSQL can contain bind variable place-holders in the form @[$column_name], where column_name is the name of a column in newRow or oldRow. If the bind variable is used in the SET clause, it will come from the newRow variable, if it is used in the WHERE clause, it will come from the oldRow variable.
- newRow is the row containing the values referenced by the bind variable place-holders of the new values, i.e. those specified in the SET clause.
- oldRow is the row containing the values referenced by the bind variable place-holders of the old values, i.e. those specified in the WHERE clause.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $update()).

$execute()

```
Do oSQL.$execute(cSQL) Returns id
```

Executes a SQL statement that does not return a result set, intended for use with DDL administrative commands such as CREATE, DROP and ALTER.

- cSQL is the SQL statement to be executed. Note that bind variable place holders are not supported.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $execute()).

The following methods may be used to obtain database meta-data:

$selecttables()

```
Do oSQL.$selecttables() Returns id
```

Retrieves table names defined in the local database.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $selecttables()).
- Single-column list containing the TableName of each table in the local database.

$selectcolumns()

```
Do oSQL.$selectcolumns(tableName) Returns id
```

Retrieves column names and type information for the specified table.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $selectcolumns ()).
- A list describing the table column definitions, defined with the following columns:
  ColumnName - name of the table column.
  SqlType - name corresponding to the column's SQL data type.
  ColumnSize - the size of a variable-length data type, e.g. for CHAR and BINARY.
  Precision - the numeric precision for a NUMERIC column. Zero for others.
  Scale - the numeric scale for a NUMERIC column. Zero for others.
  Default - the default value that was assigned to the column when the table was created.

$selectindexes()

```
Do oSQL.$selectindexes(tableName) Returns id
```

Retrieves column index information for the specified table.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $selectindexes ()).
- A list describing the table column definitions, defined with the following columns:
  IndexName - name of the index.
  ColumnNames - comma-separated list of column names used by the index.
  PrimaryKey - kTrue if the index was created with the PRIMARY KEY clause.
  Unique - kTrue if the index was created with the UNIQUE clause.

Database Synchronization

The SQL object provides two further methods that facilitate dynamic synchronization with a third-party synchronization server. The embedded UltraLite database requires connection to a Sybase Mobilink server. This is installed as part of SQLAnywhere 12.01

Please refer to the Mobilink User Guide (http://*download.sybase.com/pdfdocs/awg0901e/dbmlen9.pdf)* for guidance on:

- Setting-up the MobiLink Synchronization server.
- Configuring MobiLink users, tables and scripts.
- Creating a Consolidated Database and Installing Mobilink system tables.

The user guide also provides a useful first-steps tutorial that should prove useful. As a quick-start guide, following lessons 1-5 in the following tutorial should get you to a point at which you can test synchronization with the wrapper:

http://dcx.sybase.com/1201/en/mlstart/ml-sc-tutorial.html

$syncinit()

```
Do oSQL.$syncinit(syncParams) Returns id
```

Initializes synchronization with the (MobiLink) synchronization server. Synchronization parameters are implementation-specific and supplied via a row variable. If synchronization initialization is successful, an initial 'sync' is also performed. See $sync() for details.

The UltraLite module currently recognizes the following parameters:

Username – MobiLink synchronization user name.
Password – MobiLink synchronization user password, if required.
NewPassword – Allows the MobiLink user to change password to NewPassword, if supplied.
Version – Determines which script version to use for various synchronization actions.
Stream – Determines the network protocol to be used, e.g. *tcpip*
StreamParams – Allows protocol-specific connection parameters to be supplied, e.g. *host*
Publications – Lists the MobiLink publications to which the user is subscribed.
AdditionalParams – A string of ***name=value;*** pairs specifying any additional parameters.
Ping – If present, confirms communication with the Mobilink Server only. No synchronization occurs.
UploadOnly – If present, no changes are downloaded from the CDB, only uploads are processed.
DownloadOnly – If present, no changes are uploaded to the CDB, only downloads are received.
ResumePartialDownload – If present, UltraLite resumes a failed download only. No upload occurs.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $syncinit ()).

Example:

```
Calculate config as row(Username,Version,Stream,StreamParams)
   //define using local variables
Do
   config.$assigncols('ml_sales2','default','tcpip','host=192.168.0.
   10')
Do oSQL.$syncinit(config) Returns id
```

$sync()

```
Do oSQL.$sync() Returns id
```

Once synchronization has been initialized using $syncinit(), $sync() performs ad-hoc synchronization between the (UltraLite) database and the (MobiLink) synchronization server. If it is required to modify the synchronization parameters, $syncinit() may be called instead since this also performs synchronization on successful initialization.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $sync ()).

For Studio 6.1, sqlobject.$sync() accepts an optional parameter which can be set to 'upload' or 'download' for synchronization in one direction only (only works for SQLite).

UltraLite Database Support

To use UltraLite offline storage and Mobilink device synchronization as described above, the mobile device application is linked with the *dbUltraLite* library in addition to the JavaScript wrapper application.

SQLite Database Support

The SQLite offline storage and synchronization process uses a SQLite database on the remote client device in place of Sybase UltraLite. More specifically, SQLite synchronization relies on SQLite databases on the server and on each client device to store user tables as well as synchronization status info. The SQLite Synchronization Server uses these tables to pass data to/from each synchronization client and to forward synchronization requests on to the Consolidated Database (CDB). The SQLite Synchronization process is described in the 'SQLite Synchronization Server' manual which you can download from the Omnis website: www.tigerlogic.com/omnis/download

To use the SQLite database object in place of the UltraLite database object, the mobile device application is linked with the *dbSQLite* library in place of the *dbUltraLite* library.

Other than this, operation and use of the SQL object remains essentially the same. The SQLite initialization parameters differ slightly, as shown below.

$syncinit()

Do oSQL.$syncinit(syncParams) Returns id

The SQLite module currently recognizes the following parameters:

Username – The synchronization user name (defined at the synchronization server).
Password – The synchronization user password (defined at the synchronization server).

HostString – Omnis Web-thin Client URL to the SQLite Synchronization Server.
Timeout – The timeout in seconds for synchronization operations.

On completion, $sqldone() is called with the following parameters:

- The request id (as returned from $syncinit ()).

Example:

```
Do config.$define(Username, Password, HostString, Timeout)  ;;define
   using local variables
Do config.$assigncols(
   'user1','xxxxxx','http://192.168.0.10:7001/ultra?
   OmnisClass=rtSync&OmnisLibrary=SyncServer', 5)
Do oSQL.$syncinit(config) Returns id
```

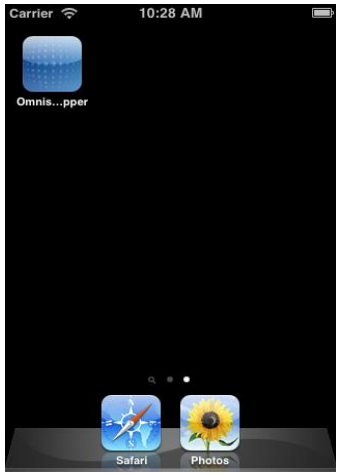
Please refer to the 'SQLite Synchronization Server' manual for information on the design, implementation and usage of the synchronization server. You can download this manual from the Omnis website (www.tigerlogic.com/omnis).

No Database Support

If the remote client application does not require database support, the application can instead be linked with the *dbNoSQL* library. This library provides stub definitions for the database API calls required by the wrapper application. Note that in this mode however, any calls to the SQL object will fail.

JavaScript Wrapper Application

The wrapper encapsulates a *WebView* which hosts the JavaScript application. The wrapper initializes using the supplied config.xml file which also informs the wrapper of the HTML page to load for the application.



The wrapper can be configured to connect to the Omnis IDE as a client for testing purposes. This is achieved via the Test Form menu option in the Studio IDE. The wrapper also provides device-independent access to features such as GPS, the camera and audio interface (see elsewhere in this manual for details about accessing device features).

To support serverless operation, the WebView runs local scripts that contain client executed methods.

Before you compile the app, you will need to customize the *config.xml* file. (There is a separate manual on the Omnis website which describes the customization of the *config.xml* file and the wrapper applications.)

When the client operates in online mode it uses the URL parameter from the config file to load the remote form and the wrapper behaves like a standard JavaScript application. If the

client is to run offline however, the other config parameters are used to allow the wrapper to (optionally) update its local copy of the application, or to run the forms locally.

iOS Wrapper Project

The iOS wrapper project now has three targets, with differing local database support. These are:

- **OmnisJSWrapper** – No local database support.
- **OmnisJSWrapper_SQLite** – SQLite is used for local database support. Synchronization with a back-end server is still under development, but is expected to be available in the next release.
- **OmnisJSWrapper_UltraLite** – UltraLite is used for local database support. Uses MobiLink for synchronization.

UltraLite

Please note that, due to licensing restrictions, we are unable to ship the libulrt.a file, on which the UltraLite target depends. As such, if you wish to build the UltraLite version of the wrapper, you will first have to compile this file yourself.

In order to do this, you should install SQLAnywhere12. In the SQLAnywhere12 installation's *ultralite/iphone* directory you will find the source code etc, and a readme file which should guide you through the build process (Please note that we offer no support for the compilation of this file).

Once you have built the *libulrt.a* (for device or simulator), you should drop it into the appropriate folder (*-iphoneos or –iphonesimulator*) in the wrapper project's *dbInterface* directory. The project should now build successfully.

Wrapper Application Source Files

The source code for the wrapper applications is no longer provided in the Omnis development tree. The projects and source code for the wrapper apps are available to download from the Omnis website: www.tigerlogic.com/omnis. The ZIP files contain template config.xml files, together with the project and source files so you can customize the wrappers if required.

There are tech notes on the Omnis website about using and customizing the wrapper applications for mobile app deployment:

- TNJS0001: "Building & Customizing the Android Wrapper App"
- TNJS0002: "Building & Customizing the iOS Wrapper App"
- TNJS0004: "Building & Customizing the BlackBerry Wrapper App"

The build process for each platform is described in the respective tech note.

Configuring the Wrapper Application

The structure for the configuration file for the wrapper applications has changed, and now a more generic format is provided for all mobile platforms. The config.xml file contains the URL for the page containing your JavaScript remote form and depending on the platform, may contain a number of other parameters (as described in the appropriate tech note, as above). The config.xml is based on the following structure:

```
<?xml version="1.0" encoding="utf-8" ?>
<settings>
   <AppTitle>0</AppTitle>

   <MenuIncludeSettings>1</MenuIncludeSettings>
   <MenuIncludeOffline>1</MenuIncludeOffline>
   <MenuIncludeAbout>1</MenuIncludeAbout>

   <SettingsFloatControls>0</SettingsFloatControls
   <SettingsScaleForm>1</SettingsScaleForm
   <SettingsAllowHScroll>0</SettingsAllowHScroll>
   <SettingsAllowVScroll>0</SettingsAllowVScroll>
   <SettingsMaintainAspectRatio>0</SettingsMaintainAspectRatio>
   <SettingsOnlineMode>1</SettingsOnlineMode>

   <ServerOmnisWebUrl>http://172.19.250.25:5911</ServerOmnisWebUrl>
   <ServerOnlineFormName>/jschtml/rfOnline</ServerOnlineFormName>
   <ServerOmnisServer></ServerOmnisServer>
   <ServerOmnisPlugin></ServerOmnisPlugin>
   <ServerOfflineFormName>rfOffline</ServerOfflineFormName>
   <ServerAppScafName>mylib</ServerAppScafName>

   <TestModeEnabled>0</TestModeEnabled>
   <TestModeServerAndPort>172.19.250.25:5911</TestModeServerAndPort>
</settings>
```

The config.xml contains the following standard properties:

- **AppTitle**
  whether or not the app displays a title bar at the top.
- **MenuIncludeSettings**
  whether the "Settings" menu option is available at runtime.
- **MenuIncludeOffline**
  whether the runtime menu option is available to switch to offline mode.
- **MenuIncludeAbout**
  whether the runtime menu's "About" option is available.

- **SettingsFloatControls**
  This property is only significant when SettingsScaleForm is "0" (false). In this case, the client uses the new $screensizefloat property of each JavaScript Client control on the form. When applying the screen size, the client uses $screensizefloat to float the edges of controls using the same rules as $edgefloat (note that the component values are not supported, just the edge-related values). If the form is wider or taller than the screen, floating only occurs if the relevant SettingsAllowHScroll or SettingsAllowVScroll parameter is false. The amount by which the controls float is the difference between the actual screen width or height and the designed width or height of the form for the closest matching $screensize. The value of $screensizefloat is stored for each setting of $screensize in the remote form
- **SettingsScaleForm**
  If you set this to "1" (true), the client scales the form to fit the available screen space. The scaling factor is the screen width or height divided by the width or height of the closest matching $screensize. For these purposes, the actual screen size excludes the operating system areas such as the status bar
- **SettingsAllowHScroll** and **SettingsAllowVScroll**
  set these to "1" if you want to allow horizontal or vertical scrolling of the form respectively, or "0" if not
- **SettingsMaintainAspectRatio**
  If you set this to "1", scaling maintains the aspect ratio of the form. When turned on, and depending on SettingsAllowHScroll and SettingsAllowVScroll, it may reduce the scaling factor in one direction, to make the form fit, and center the form vertically or horizontally as required
- **SettingsOnlineMode**
  whether the app starts in Online mode.
- **ServerOmnisWebUrl**
  URL to the Omnis or Web Server. If using the Omnis Server it should be http://<ipaddress>:<omnis port>. If using a web server it should be a URL to the root of your Web server. http://myserver.com
- **ServerOnlineFormName**
  route to the form's .htm file from ServerOmnisWebUrl. So if you're using the built in Omnis server, it will be of the form /jschtml/myform.htm. If you are using a web server, it will be the remainder of the URL to get to the form, e.g. /omnisapps/myform. (Do not add the .htm extension!)

Only ServerOmnisWebUrl & ServerOnlineFormName are needed for Online forms. The other Server… properties are for Offline mode.

- **ServerOmnisServer**
  The Omnis Server <IP Address>:<Port>.
- **ServerOmnisPlugin**
  If you are using a web server plug-in to talk to Omnis, the route to this from ServerOmnisWebUrl. E.g. /cgi-bin/omnisapi.dll

- **ServerOfflineFormName**
  Name of the offline form. (Do not add .htm extension!)
- **ServerAppScafName**
  Name of the App SCAF. This will be the same as your library name.
- **TestModeEnabled**
  Enable test mode (Ctrl-M on form from Studio to test on device)
- **TestModeServerAndPort**
  the <ipaddress>:<port> of the Omnis Studio Dev version you wish to use test mode with.

You can also change these parameters by pressing the menu button on the mobile device, and using the menu options to change them. The app remembers the last setting made via the menu, so the config.xml lets you set the initial values in the wrapper application.

Testing Remote Forms in a Wrapper App

During development, you can open a JavaScript remote form in a wrapper application using the *Test Form Mobile* (Ctrl-M) option, assuming a wrapper application is setup and enabled (otherwise you can still test your mobile forms in a desktop browser on your development computer before you setup the wrapper app). This option appears beneath the 'Test Form' option in the remote form context menu. The Test Form Mobile option is only displayed in the relevant menus when both:

1. A wrapper application is enabled for test form (see the menu of the Android app, and the system settings for the iOS app)
2. A wrapper application is connected to the Omnis App Server, using the test form parameters.

The $designshowmobiletitle property determines whether or not the title of a wrapper application is visible when you use the Test Form Mobile (Ctrl-M) option. For deployment, config.xml allows you to configure whether or not the title is displayed in the wrapper application.

Accessing Mobile Device Features

The new JavaScript Client wrapper application allows your mobile apps to access hardware and software based services on the end user's mobile device, such as a smartphone or tablet. Access to these mobile device features is available via a new JavaScript component called the **Device Control,** available in the Component Store when you design a JavaScript based remote form. The Device control itself is invisible and to enable access to the device functionality you need to add the Device Control to your remote form and assign an action to the $action property of the control at runtime using methods.

The new Device Control allows access to mobile device features such as getting the location of the end user's device using GPS, retrieving the contacts information from the device, or returning images from either the camera or photo library on the device. Depending on the type of application you are creating, some or all of these features may be useful to enhance the interactivity and functionality of your app for end users when they run your app on a mobile device, such as a smartphone or tablet computer.

Running the Device control and Compatibility

Important Note *the Device Control only works in an application that is running within one of the wrapper applications on a mobile device;* it will not work if the app is run in a standard web browser on a mobile device (or a desktop computer). Therefore you will need to create your mobile app, compile it into a wrapper app, and test it in a simulator or directly on a mobile device by running the native app. The new wrapper app that supports the Device control is available for iOS, Android, and BlackBerry.

The Device control supports a number of hardware functions, some of which may not be available on particular devices. You should test your app thoroughly on the specific devices you wish to support with each of the device functions that you want end users to access.

For some of the hardware features, Omnis can detect if they are not present on the current mobile device running the app. For example, if a device does not have a hardware camera then the action kJSDeviceActionTakePhoto will report an evPhotoFailed message.

Properties

Note the Device control is invisible, therefore some of the visual properties normally associated with JavaScript components may not be relevant, such as $alpha.

Property	Description
$action	The "action" for the Device control which specifies which function on the client mobile device is accessed; this is assigned as a constant, as follows: kJSDeviceActionBeep, kJSDeviceActionGetBarcode, kJSDeviceActionGetContacts, kJSDeviceActionGetGps, kJSDeviceActiongetImage, kJSDeviceActionMakeCall, kJSDeviceActionSendEmail, kJSDeviceActionSendSms, kJSDeviceActionTakePhoto, kJSDeviceActionVibrate
$barcodeimage	If true, a barcode image will be returned.
$communicationaddress	Contains contact address details when the Make Call and Send SMS device actions are used.
$communicationdata	Contains the data to be sent when the Send SMS device action is used.
$dataname	The name of an instance variable for the Device control
$deviceimage	Contains an instance variable name used for holding the image returned from the device.
$soundname	Name of the sound sample to be played when the Beep device action is called.
Standard	$align $alpha $backalpha $backcolor $bordercolor $componentctrl $componentlib $contextmenu $dataname $disablesystemfocus $edgefloat $effect $enabled $events $fieldstyle $font $fontsize $height $ident $left $linestyle $name $objtype $order $screensizefloat $tooltip $top $userinfo $visible $width

Contact properties

The following properties are relevant when the kJSDeviceActionGetContacts device action is used. By setting these properties you can control what information is returned from the Contacts data on the device.

Property	Description
$contactaddresses	If true, device contact requests will include contact's address.
$contactbirthday	If true, device contact requests will include contact's birthday.
$contactcategories	If true, device contact requests will include categories info.
$contactdisplayname	If true, device contact requests will include the display name.
$contactemails	If true, device contact requests will include email info.
$contactims	If true, device contact requests will include ims info.
$contactname	If true, device contact requests will include name info.
$contactnickname	If true, device contact requests will include nick name info.
$contactnotes	If true, device contact requests will include notes info.
$contactorganization	If true, device contact requests will include organization info.
$contactphonenumbers	If true, device contact requests will include phone number info.
$contactphotos	If true, device contact requests will include photo images.
$contacturls	If true, device contact requests will include url info.

Events

Event	Description
evBarcodeFailed	Sent when no Barcode could be obtained from the device. **Parameters** pEventCode: The event code
evBarcodeReturned	Sent to the device control when a Barcode is ready for processing. **Parameters** pDeviceBarcode: The Barcode data pEventCode: The event code
evContactsFailed	Sent when no contacts could be obtained from the device. **Parameters** pEventCode: The event code
evContactsReturned	Sent to the device control when contacts information is ready for processing. **Parameters** pEventCode: The event code
evGpsReturned	Sent to the device control when Location Data is ready for processing. **Parameters** pDeviceGps: The GPS location pEventCode: The event code
evImageFailed	Sent when the device failed to return an image. **Parameters** pEventCode: The event code
evImageReturned	Sent to the device control when an image is ready for processing. **Parameters** pDevicePhoto: Image from Camera pEventCode: The event code
evPhotoFailed	Sent when the device failed to take a photo. **Parameters** pEventCode: The event code
evPhotoReturned	Sent to the device control when an image is returned from camera ready for processing. **Parameters** pDevicePhoto: Image from Camera pEventCode: The event code
Standard	evExecuteContextMenu evOpenContextMenu

Setting the Action Property

The following list summarizes the actions available and the constant needed for the $action property:

- **kJSDeviceActionBeep** – Beep Device
  forces the device to play the default beep
- **kJSDeviceActionGetBarcode** – Get a Barcode or QR code
  returns the output from Barcode/QR-code scanning function on the device (if available); the output is usually a string which can be a URL
- **kJSDeviceActionGetContacts** – Get contact info
  returns contact information from the device; note there are other properties to determine the content or extent of the contact information returned
- **kJSDeviceActionGetGps** – Get the device location
  returns the location data using the GPS function on the device
- **kJSDeviceActionGetImage** – Get an Image
  returns an image from the device's image gallery
- **kJSDeviceActionMakeCall** – Make a Phone Call
  forces the device to make a phone call (if available)
- **kJSDeviceActionSendEmail** | **kJSDeviceActionSendSms** – Send an Email or SMS
  forces the device to send an Email or SMS / text message (if available)
- **kJSDeviceActionTakePhoto** – Take a Photo
  forces the device to take a photo (if a camera is available)
- **kJSDeviceActionVibrate** – Vibrate the Device
  forces the device to vibrate (if available)

The basic method to assign an action to the Device control is as follows:

```
Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceAction…)
; where oDevice is the name of the Device control
```

See the *Reference* section at the end of this manual for a full list of properties and events for the Device control.

Beep Device Action

To make the device play a given sound sample, you need to assign the constant kJSDeviceActionBeep to the $action property. This is one way communication with the device which will result in the device playing a sound sample. To specify which sound to play, you need to set the $soundname property to the name of the sound sample to be played which must be complied into the wrapper application. The wrapper contains a default sound called "Notify".

Example

```
On evClick
   Calculate $cinst.$objs.oDevice.$soundname as "Notify"
   Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionBeep)
```

Get Barcode Device Action

You can return a Barcode or QR code by assigning the constant kJSDeviceActionGetBarcode to the $action property. If the action is successful an evBarcodeReturned event is sent to the Device Control and the barcode data is returned in the pDeviceBarcode event parameter; the barcode data is usually a string containing Alphanumeric characters, such as a product number or name, or in the case of a QR code it could be a website URL.

Example

```
; event method for "Scan" button
On evClick
   Do
   $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionGetBarcode)
```

The event method for the Device Control could be:

```
On evBarcodeReturned
   Do
   iProducts.$search(iProducts.iProdQrCode=pDeviceBarcode,kTrue,kFal
   se,kFalse,kFalse)
   If iProducts.$line=0
      Do iProducts.$line.$assign(iProducts.$linecount)     ;; other
   End If

   Do iProducts.$loadcols()
   Calculate iAmount as 1

   If iProdName='Other'
      Calculate iProdName as pDeviceBarcode     ;; show the value of
   the barcode
   End If
```

Vibrate Device Action

To make the device vibrate you need to assign the constant kJSDeviceActionVibrate to the $action property. This is one way communication with the device which will result in the device vibrating for a short period of time.

Example

```
On evClick
   Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionVibrate)
```

Get GPS Device Action

To receive location (GPS) data from the device you need to assign the constant kJSDeviceActionGetGps to the $action property. The evGpsReturned event is sent when the location data has successfully been returned. The event parameter pDeviceGps will contain the returned data which is formatted as a string containing longitude and latitude data separated by a colon ":". If the device fails to obtain location data or the device does not support location tracking, the returned data will be a longitude and latitude of zero, i.e. "0.0:0.0".

Example

```
On evClick
   Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionGetGps)
On evGpsReturned
   Calculate $cinst.$objs.oMap.$latlong as pDeviceGps
```

Take Photo / Get Image Device Action

To take a photo with the device (if a camera is present) or to return an image from the device's gallery the kJSDeviceActionTakePhoto or kJSDeviceActionGetImage constants need to be assigned to the $action property. The $dataname property of the Device component needs to be assigned to an Instance Variable of type Binary to hold the incoming image from the device. If the device is successful in returning an image, the $event evPhotoReturned or evImageReturned will be called to indicate that the operation has been successful and the instance variable in $dataname will contain base64 encoded image data. In the event of the device failing to return an image or the user cancels the request, the event evPhotoFailed or evImageFailed will be sent.

Example

```
On evClick
   Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionTakePhoto)
```

In this case the instance variable iImage is a Binary variable. The $deviceimage property is set to iImage. There is another Binary variable called ipic which is associated to a picture component. By copying the returned image in iImage into the picture component variable you can display the image returned from the device.

```
On evPhotoReturned
   Calculate iPic as iImage
```

Get Contacts Device Action

To obtain information from the contacts database on the device the kJSDeviceActionGetContacts constant must be assigned to the $action property. To accommodate the contact database the $dataname property needs to be assigned to an Instance Variable of type List. The properties starting with $contact... determine which contact fields will be obtained from the device: setting these properties to true or false will determine if that specific field is returned from the device.

The evContactsReturned event is triggered when the contact database has been returned, and in the case of the device failing to obtain the contact database the evContactsFailed event is triggered.

Example

```
On evClick
   Do $cinst.$objs.oDevice.$action.$assign(
           kJSDeviceActionGetContacts)
; retrieve info from the Contact list
On evContactsReturned
   Set reference lNameRow to iDeviceList.name.1
   Calculate iNameRow.FirstName as lNameRow.givenName
   Calculate iNameRow.MiddleName as lNameRow.middleName
   Calculate iNameRow.Surname as lNameRow.familyName
   Calculate iNameRow.Prefix as lNameRow.honorificPrefix
   Calculate iNameRow.Suffix as lNameRow.honorificSuffix
   Calculate iNameRow.Nickname as iDeviceList.nickName
```

Contacts data structure

- **displayName:** The name of this Contact, suitable for display to end-users (String).
- **name:** An object containing all components of a person's name.
  - **formatted:** The complete name of the contact (String).
  - **familyName:** The contacts family name (String).
  - **givenName:** The contacts given name (String).
  - **middleName:** The contacts middle name (String).
  - **honorificPrefix:** The contacts prefix (example Mr. or Dr.) (String).
  - **honorificSuffix:** The contacts suffix (example Esq.) (String).
- **nickname:** A casual name to address the contact by (String).
- **phoneNumbers:** a list of all the contact's phone numbers.
  - **type:** A string that tells you what type of field this is (example: 'home') (String).
  - **value:** The value of the field (such as a phone number or email address) (String).
  - **pref:** Set to true if this ContactField contains the user's preferred value (Boolean).
- **emails:** a list of all the contact's email addresses.
  - **type:** A string that tells you what type of field this is (example: 'home') (String).
  - **value:** The value of the field (such as a phone number or email address) (String).
  - **pref:** Set to true if this ContactField contains the user's preferred value (Boolean).
- **addresses:** a list of all the contact's addresses.
  - **pref:** Set to true if this ContactAddress contains the user's preferred value (Boolean).
  - **type:** A string that tells you what type of field this is (example: 'home') (String).
  - **formatted:** The full address formatted for display (String).
  - **streetAddress:** The full street address (String).
  - **locality:** The city or locality (String).

region: The state or region (String).
postalCode: The zip code or postal code (String).
country: The country name (String).

- **ims:** a list of all the contact's IM addresses.
  type: A string that tells you what type of field this is (example: 'home') (String).
  value: The value of the field (such as a phone number or email address) (String).
  pref: Set to true if this ContactField contains the user's preferred value (Boolean).
- **organizations:** a list of all the contact's organizations.
  pref: Set to true if this Contact organization contains the user's preferred value (Boolean).
  type: A string that tells you what type of field this is (example: 'home') (String).
  name: The name of the organization (String).
  department: The department the contract works for (String).
  title: The contacts title at the organization (String).
- **birthday:** The birthday of the contact (Character).
- **note:** A note about the contact (String).
- **photos:** a list of the contact's photos.
  type: A string that tells you what type of field this is (example: 'home') (String).
  value: The value of the field (such as a phone number or email address) (String).
  pref: Set to true if this ContactField contains the user's preferred value (Boolean).
- **categories:** a list of all the contacts user defined categories.
  type: A string that tells you what type of field this is (example: 'home') (String).
  value: The value of the field (such as a phone number or email address) (String).
  pref: Set to true if this ContactField contains the user's preferred value (Boolean).
- **urls:** a list of web pages associated to the contact.
  type: A string that tells you what type of field this is (example: 'home') (String).
  value: The value of the field (such as a phone number or email address) (String).
  pref: Set to true if this ContactField contains the user's preferred value (Boolean).

Make a Call Device Action

To make a phone call from the device the kJSDeviceActionMakeCall constant is used. Before assigning this action the phone number for the call should be specified in the $communicationaddress property.

Example

```
Do $cinst.$objs.oDevice.$communicationaddress.$assign("0123456789")
Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionMakeCall)
```

Send an SMS Device Action

To send an SMS (Short Message Service) from the device the kJSDeviceActionSendSMS constant is used. Before assigning this action the phone number for the SMS should be specified in the $communicationaddress property, and contents of the message should be specified in $communicationdata.

Example

```
Do $cinst.$objs.oDevice.$communicationaddress.$assign("0123456789")
Do $cinst.$objs.oDevice.$communicationdata.$assign("A message")
Do $cinst.$objs.oDevice.$action.$assign(kJSDeviceActionSendSMS)
```

Resizable Forms and Components

This version introduces one new property ($dragborder) and modifies another ($edgefloat) to allow JavaScript components and remote forms to be dynamically resizable at runtime in the end user's browser. In addition, remote forms have the $resizemode property which allows a form displayed in a desktop web browser to be resized.

Web Form Resizing

JavaScript remote form classes now have a property called $resizemode which determines whether or not the form resizes when the end user resizes the browser window. In previous versions, the size of a remote form was fixed regardless of the size of the browser window displaying the form or whether or not the browser window was resized, but using the new property you can make the form resize if required.

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

The value of $resizemode is one of the kJSformResizeMode... constants that specify how the form behaves when it initially opens and when the browser window is resized. The kJSformResizeMode... constants are as follows:

- **kJSformResizeModeNone**
  The form does not change size when the browser window is resized and the form is positioned at the left of the browser window. This corresponds to the behaviour in Omnis Studio 5.2.x
- **kJSformResizeModeCenter**
  The form does not change size when the browser window is resized but the form is centred horizontally in the browser window, only if its width is less than the browser window width
- **kJSformResizeModeAspect**
  The form resizes itself as the browser window is resized maintaining its aspect ratio to fit the browser window; it will not resize to a size smaller than the designed size in the remote form class
- **kJSformResizeModeFull**
  The form resizes itself to fit the browser window, regardless of aspect ratio; it will not resize to a size smaller than the designed size in the remote form class

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

Existing users should note that the omnisobject parameter "data-screensize", which performed a similar function to $resizemode and was only available for testing purposes, has been removed from the client. Use $resizemode to control web form resizing.

Component Resizing

The "floating edge" ($edgefloat) capabilities for JavaScript components have been modified in this version, so *when a remote form is resized at runtime* (in the end user's browser) its components will now resize automatically, if their $edgefloat property has been set as appropriate.

In previous versions, the $edgefloat property only applied to resizing components in design mode, but now the setting of $edgefloat for each component is applied at runtime when the form is resized on the client. The $edgefloat property can be set to one of the kEF… constants which determines which edges of the component, if any, will "float" when the form is resized. See the Omnis Help (F1) for a list of all possible settings of $edgefloat.

You can store a different setting of the $edgefloat property for each component, for each different screen size ($screensize), which means you can set different $edgefloat properties for web and mobile browsers. When setting $edgefloat in the Property Manager in design mode, you can set the value of $edgefloat for all $screensize values by holding the Control key when selecting the $edgefloat value.

The setting of $edgefloat for a component is used to resize the component (or not if set to kEFnone) when the form or container field is resized at runtime, and when one or more of the following occurs:

- When the component is in a subform and the subform is resized (that is, its size at runtime is different to the size of the subform class)
- When applying a different mobile device size while running in a mobile device custom wrapper
- When the $resizemode property of the form causes the form to resize
- When the component is in a resizable subform in a subform set and the subform is resized

$edgefloat and $screensizefloat compatibility

Existing users who have used $screensizefloat in a standalone app deployed using the Custom Wrappers should note the following.

The $screensizefloat property has been removed, and its functionality has been merged with the $edgefloat property that now supports component floating edges at runtime in the browser.

When converting a library from Omnis Studio 5.2.x to 6.0, the value of $edgefloat becomes either:

- none if both values of $edgefloat and $screensizefloat are none

- or whichever of the two values is not none
- or in the very unlikely case that both these properties have a value different to none, the value of $screensizefloat is used (the $edgefloat value is discarded)

For compatibility, you can still use $screensizefloat in notation code, but it maps directly to $edgefloat.

Draggable Component Borders

End users can now resize some JavaScript components dynamically at runtime in their web browser. When the end user's mouse is over the edge of a component that can be resized, the cursor changes to indicate that the border can be dragged and resized.

To allow this functionality, JavaScript components have a new $dragborder property, which only applies when a component has its $edgefloat property set to one of the kEFposn… constants (other than kEFposnClient or kEFposnJoinHeaders). If $dragborder is set to true, and you have set $edgefloat as above, the end user will be able to resize the component at runtime by dragging the border of the component with the mouse.

You can store a different setting of the $dragborder property for each component, for each different screen size ($screensize), therefore components on the same form could be resizable for web desktop browsers and not for mobile devices. When setting $dragborder in the Property Manager in design mode, you can set the value of $dragborder for all $screensize values by holding the Control key when selecting the $dragborder value.

The appearance of the drag border area can be modified by editing the styles div.omnis-db-vert and div.omnis-db-horz in omnis.css.

Subform Sets

You can now open a special kind of subform or group of subforms that behave rather like separate windows in the JavaScript Client. The new subforms are different to standard subforms in that they have a title bar and resizable borders, so the end user can move or resize them dynamically within the "main" JavaScript remote form running on the client. This will allow you to create highly flexible user interfaces in your web and mobile apps, by allowing a high degree of interactivity for the end user.

The new subforms or group of subforms are opened as part of a new client object called a **Subform Set** (SFS), which is created at runtime in the JavaScript Client and allows you to manage the group of subforms. The new subforms are opened in the client at runtime within the main JavaScript remote form, or they can be opened within the context of a single page in a paged pane. Each separate subform in the Subform Set is a standard Remote form class that you have previously created in your library, which is referenced and added to the Subform Set.

Stacking Order List

The subforms in a Subform Set have a "stacking order" (or Z-order) relative to one another, so the top-most form in the set will appear in front of any forms lower in the stacking order

if they intersect. Clicking on a form lower in the stacking order brings it to the front. The tab order of the remote form excludes controls on the subforms in a Subform Set behind the top form in the set. There is a maximum of 256 remote form instances (including subform instances) in a remote task instance. When you have more than one Subform Set open (this is allowed but not recommended) there is no relative stacking order between the sets.

Creating Dynamic Subforms

There is a set of new client commands to open and manage the subforms in a Subform Set which you can execute using the $clientcommand() method. These client commands must be executed on the Omnis App Server in the context of the current remote form instance ($cinst); the $clientcommand() method will not work in a client-side method. The $clientcommand() method requires two parameters: the *cCommand* to be executed and a *wRow* variable containing the parameters for the command, with the syntax:

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

where $cinst is the current remote form instance.

Subform Client Commands

The following client commands are available for creating and managing Subform Sets.

subformset_add

The **subformset_add** command creates a Subform Set within the current remote form instance.

```
Do $cinst.$clientcommand("subformset_add",row-variable)
```

Where *row-variable* is row(**setname, parent, flags, ordervar, formlist**)

Note the parent parameter is available if you want to create the subform set inside a paged pane, rather than the remote form instance. The columns for the row variable parameter are as follows:

setname: a string which is the name of the set, which must be unique within the current remote task.

parent: the container for the set, either:

- pagedpanename:page (e.g. pp:5), so that the subforms belong to the specified page of the paged pane)
- or empty, meaning that the subforms in the set belong to the remote form instance invoking $clientcommand.

flags: the sum of 0-4 of the following constants, which affect the behavior of the forms in the set:

- kSFSflagCloseButton: The subforms in the set have a close button
- kSFSflagMinButton: The subforms in the set have a minimize button
- kSFSflagMaxButton: The subforms in the set have a maximize button; note the subform can only be maximized (resized) if kSFSflagResize is enabled

- kSFSflagResize: The subforms in the set have a resize border so that they can be resized using the mouse or when the Maximize button is pressed

ordervar: the name of an instance list variable in the remote form invoking the $clientcommand. The client keeps this list variable up to date with the stacking order and position information for the subforms in the set: see below.

formlist: a list which defines the subforms to be added initially to the subform set (this list can be empty), i.e. a list of remote form classes that you have previously created in your library. The order of the forms in this list represents the stacking order from top to bottom, so that once the set has been added, the top-most subform will be for line 1, and the bottom-most subform will be for the last line. The columns in the list are as follows:

- Column 1: uniqueID: An integer which must uniquely identify this subform in the set.
- Column 2: classname: The name of the Omnis remote form class for the subform.
- Column 3: params: Literal parameters to be passed to the $construct of the subform e.g. 'Test',200.
- Column 4: title: The title of the subform - text displayed in the title bar of the subform.
- Column 5: left: The left coordinate of the subform (for Desktop browsers, or portrait if a mobile device). The constant kSFScenter centers the form horizontally in its parent.
- Column 6: top: The top coordinate of the subform (for Desktop browsers, or portrait if a mobile device). The constant kSFScenter centers the form vertically in its parent.
- Column 7: width: The width of the subform. If the forms in the set are resizable, then the form cannot be made narrower than the minimum of this width and the width designed for the remote form class.
- Column 8: height: The height of the subform. If the forms in the set are resizable, then the form cannot be made taller than the minimum of this height and the height designed for the remote form class.
- Columns 9-12: If the subforms are to be displayed on a mobile device, Columns 9-12 are landscape left, top, width and height respectively. If these are omitted, the landscape values default to the portrait values.

subformset_remove

The **subformset_remove** command removes a set of subforms. All subforms in the set will be destructed and removed from their parent.

```
Do $cinst.$clientcommand("subformset_remove",row-variable)
```

Where *row-variable* is row(**setname**) where **setname** is set to be removed.

subformset_formadd

The **subformset_formadd** command add a form to an existing subform set.

```
Do $cinst.$clientcommand("subformset_formadd",row-variable)
```

Where *row-variable* is row(**setname, uniqueID, classname, params, title, left, top, width, height, modal**)

The row variable parameter are as follows:

setname: a string which is the name of the set to which the subform is to be added.

uniqueID: an integer which must uniquely identify this subform in the set.

classname: the name of the Omnis remote form class for the subform.

params: literal parameters to be passed to the $construct of the subform e.g. 'Test',200.

title: the title of the subform - text displayed in the title bar of the subform.

left: the left coordinate of the subform (for Desktop browsers, or portrait if a mobile device). The constant kSFScenter centers the form horizontally in its parent.

top: the top coordinate of the subform (for Desktop browsers, or portrait if a mobile device). The constant kSFScenter centers the form vertically in its parent.

width: the width of the subform. If the forms in the set are resizable, then the form cannot be made narrower than the minimum of this width and the width designed for the remote form class.

height: the height of the subform. If the forms in the set are resizable, then the form cannot be made taller than the minimum of this height and the height designed for the remote form class.

modal: zero if the subform is non-modal, or 1 if the subform is fully modal, and prevents the use of any other form or subform in the remote task's user interface.

If the subforms are to be displayed on a mobile device, the next four columns are landscape left, top, width and height respectively. If these are omitted, the landscape values default to the portrait values.

The parameters above, starting with uniqueID, are identical to those in the formlist (for the subformset_add command), except the modal indicator is present between the two sets of coordinates.

subformset_formremove

The **subformset_formremove** command removes a subform from an existing set and destructs it (removing it from its parent).

```
Do $cinst.$clientcommand("subformset_formremove",row-variable)
```

Where *row-variable* is row(**setname, uniqueID, focus**)

The row variable parameter are as follows:

setname: a string which is the name of the set from which the subform is to be removed.

uniqueID: an integer which identifies the subform in the set to be removed.

focus: optional (default value is kFalse). If the focus parameter is kTrue, sets focus to the new top form in the set unless it is minimized.

subformset_formtofront

The **subformset_formtofront** command brings a subform in a set to the top of the stacking order, and gives it the focus. You must use this command to display a subform that has previously been minimized.

```
Do $cinst.$clientcommand("subformset_formtofront",row-variable)
```

Where *row-variable* is row(**setname, uniqueID**)

setname: a string which is the name of the set containing the subform.

uniqueID: an integer which identifies the subform in the set to be brought to the front.

Using the Stacking Order Variable (ordervar)

When you use the subformset_add client command a list called ordervar is created containing a list of the subforms in the subform set. The ordervar variable allows you to manage the subforms in the set. It has the same definition as the formlist, and like the formlist it contains the subforms in the order of the top to the bottom of the stacking order. Note that if coordinates have been centered using kSFScenter, the ordervar contains their actual values rather than the value kSFScenter.

Whenever the stacking order changes, or a form is moved or resized, the client updates the values in ordervar. This results in:

- Automatic updates to controls which are data-bound to the ordervar.
- A call to a client method in the container form for the SFS. If you add a client-executed method called $sfsorder, with a single parameter, which is the set name, you can add processing that occurs each time the set is updated. For example, you could use a tab control to display a tab for each member in the set, where the current tab represents the top-most subform.

You can use the ordervar list in conjunction with the subformset_formtofront $clientcommand to manage the subforms in the set, e.g. bring a form to the front by selecting a line in a popup menu (this is the only way to restore a minimized subform). For example:

```
; the ordervar list is assigned to iOpenForms, C1 contains the
   subform ID
Do $cinst.$clientcommand("subformset_formtofront",
   row('SubformSet',iOpenForms.[pLineNumber].C1))
```

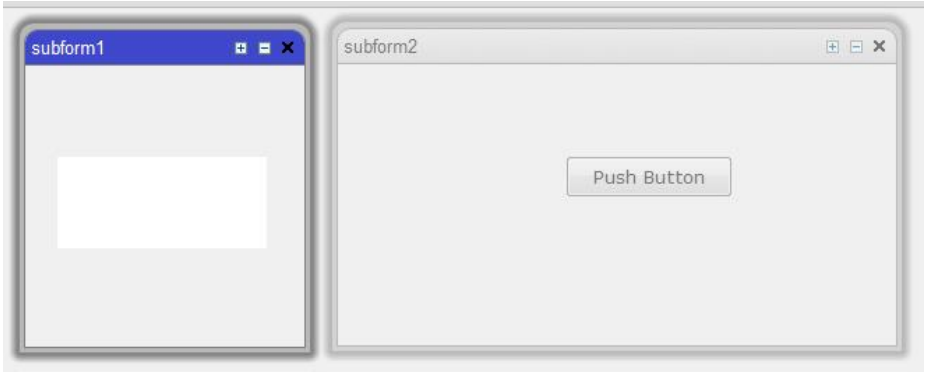
If a subform has been minimized, you would you have to use such a method to display the subform again since minimized subforms are not visible in the parent remote form.

Example

The following code creates a subform set containing two subforms.

```
; setupSubformSet method which could be called from £construct
; Create vars: iFormList (List), iID, iClassName, iParams, iTitle,
   iLeft, iTop, iWidth, iHeight
; jsSub1 and jsSub2 are remote forms in the library
Do iFormList.$define(
        iID,iClassName,iParams,iTitle,iLeft,iTop,iWidth,iHeight)
Do iFormList.$add(1,"jsSub1",,"subform1",10,10,200,200)
Do iFormList.$add(2,"jsSub2",,"subform2",220,10,400,200)
Do lRow.$define(lSetName,lParent,lFlags,lOrderVar,iFormList)
Do lRow.$assigncols(
        "SubformSet",,kSFSflagCloseButton+kSFSflagMaxButton+
        kSFSflagMinButton+kSFSflagResize,,iFormList)
Do $cinst.$clientcommand("subformset_add",lRow)
```

The code creates the subforms in the main remote form within the browser:



Subform Styles

The Omnis style sheet Omnis.css contains CSS classes that specify the appearance of the subform frame of the members of subform sets, as well as the images for the maximize and minimize buttons. You can modify these classes to give the subform frames your own style. Open Omnis.css and search for “subform” to locate the styles.

Subform Titles

You can use $cinst.$title to change the title text for a member of a subform set.

Subform References

You can obtain a reference to any subform instance within a subform set using the $sfsmember root notation. For example:

```
$root.$sfsmember(cSetname,iUniqueID)
```

returns an item reference to the remote form instance for the subform set member with the specified unique ID in the named subform set in the current remote task. This notation can be used in server and client methods.

SQL Multi-tasking and SQL Workers

In Omnis Studio 6.0, you can execute long-running tasks such as a SELECT statement on a separate background thread that reports back to the main thread as each task completes. To enable this functionality, the Omnis DAMs allow the creation of "SQL Workers" which are instantiated from a new SQL Object variable type available in the Oracle, ODBC, JDBC, MySQL, PostgreSQL, DB2, Sybase, and SQLite DAMs.

SQL Worker object completion methods allow list fields and other form data to be populated asynchronously, making applications more responsive and potentially faster where multiple SQL Workers are used.

Overview

The SQL Worker Objects support three primary methods:

- **$init()**
  Initializes or resets a worker object ready to perform its task
- **$start()**
  Starts the worker task on a background thread (non-blocking)
- **$cancel()**
  Aborts a worker task running on a background thread

There are additional properties to allow a running task to be discarded in place of a new task and to cancel such tasks as they become "orphaned". There is also a property to report the state of a worker object's running background thread.

Worker objects are created by sub-classing an Omnis Object class with the appropriate SQL Worker Object type. You initialize the object by supplying a SQL statement along with any bind variables that the SQL statement may need. Logon details or optionally the name of a session pool are also passed during initialization.

A SQL Worker thread is dispatched by calling $start(). Upon completion, the worker thread calls back into the worker object's $completed() method, or $cancelled(), with the result set or error information. Therefore, worker object completion methods may be used to populate list fields and other form data asynchronously, making applications more responsive and potentially faster where multiple worker objects are used.

SQL Worker Object Methods

Method	Description
$init()	$init(ParamRow). Initializes or resets a worker object ready to perform a unit of work.*
$start()	Starts the worker task running on a background thread (non-blocking).*
$run()	Starts the worker task running on the caller thread (blocks until complete). Intended for testing purposes.*
$cancel()	Aborts a worker task running on a background thread.*
$sessionref()	$sessionref(ObjectRef). Returns a reference to the session object being used by the underlying Worker Delegate.*
$completed()	Called by the Worker Delegate upon completion of its work.
$cancelled()	Called by the Worker Delegate if the running background task was cancelled.

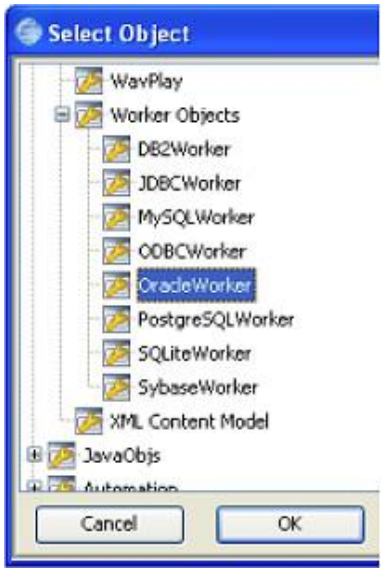
*Method returns kTrue on successful execution, kFalse otherwise.

SQL Worker Object Properties

Property	Description
$cancelifrunning	If kFalse (the default), orphaned background tasks run to completion. If kTrue, they are instructed to cancel before being detached.
$waitforcomplete	If kTrue (the default), the Interface Object waits for completion of a running background task before the object can be used again. If kFalse, the running task is detached and a new Worker Delegate takes its place.
$state	Returns the current state of the underlying background task; either kWorkerStateCancelled, kWorkerStateClear, kWorkerStateComplete, kWorkerStateInit or kWorkerStateRunning.
$errorcode	On failure of a command function, contains the error code.
$errortext	On failure of a command function, contains the error message.
$threadcount	Reports the number of worker threads currently being monitored by the underlying Thread Timer object.

Creating SQL Worker Objects

Worker objects are created by sub-classing an Omnis object class as a Worker Object. For example, using the Select Object dialog, assigning a $superclass for use with an Oracle connection results in: ***.ORACLE8DAM.Worker Objects\OracleWorker***.



To access worker functionality from your code, you then create one or more object instance variables of subtype <your-object >.

Worker Object Initialization

A worker object must be initialized on the caller thread before it can run. You initialize the object by supplying a SQL statement along with any bind variables that the SQL statement may require. Logon details or optionally- the name of a session pool are also passed during initialization.

The initialization parameters are supplied to the $init() method via a row containing attribute values. Attribute names appear in the column headings. The attribute names recognized by the Worker Object are as follows (case-insensitive):

Attribute name	Attribute value
session	A session object or object reference. The session must be logged-on and in a useable state.
poolname	The name of an existing session pool. The worker will take a session object from this pool, returning it upon completion.
hostname	The hostname/IP address of the database server.
database	The database name to use for a logon.
username	The username to use for a logon.
password	The password to use for a logon.
query	The SQL statement to be executed by the worker object.
bindvars	A list or row containing bind variable values. Bind variable names must match column names in this list/row. If the list contains multiple rows, the query is re-executed for each row.

If the *session* attribute is supplied, the other logon attributes, i.e. hostname, database, username & password are ignored, since it is assumed that the session object is already in a useable state. **Please Note:** In this mode, the session should be considered reserved for use exclusively by the worker. If the main application attempts to share a session object being used by a worker running on another thread, the results are undefined.

The logon parameters are also ignored if the poolname attribute is supplied. In this mode, the worker attempts to obtain a session from the named session pool, releasing it when the worker task completes. (If both session and poolname are supplied, poolname is ignored.)

Where neither, session or poolname are supplied, an internal session object is created dynamically. Valid logon credentials should be supplied via hostname, username and password. Although read during the call to $init(), the worker will not attempt to logon until the $run() or $start() method is called. In this mode, the session is automatically logged-off when the worker task completes (or is cancelled). Should you need to modify one or more session attributes before calling $run() or $start(), it is possible to obtain a reference to the session object by calling the worker object's $sessionref() method, for example:

```
Do iWorkerObj.$sessionref(lObjRef) Returns #F
Do lObjRef.$port.$assign(5435)
```

The SQL text supplied via the query attribute may contain any SQL statement but ideally, should be a statement that normally takes an appreciable amount of time to execute, for example; a SELECT, UPDATE or DELETE statement. The query text may also contain one or more bind variables, specified using standard @[…] notation.

Bind variable values are supplied via a separate *bindvars* parameter. The supplied list is stored during $init() and read when the worker task starts. Where the list contains multiple rows, the worker re-executes the supplied SQL statement for each row of bind variables. As of Studio 6.0.1, bind variables in the query text must reference column names in the supplied list, e.g. @[lRow.colID] or @[colID].

Running the Worker Object

The $start() method causes the worker task to run on a background thread. Thus, return from $start() is immediate and the main application is free to continue processing. For example, the iWorkerObj var has been created from the oPostgreSQLWorker class:


```
;; iWorkerObj is an instance of an Object class
Calculate Params as row(
        iSQLText,'192.168. 0.10',iUser,iPassword,iDBName)
Do Params.$redefine(query,hostname,username,password,database)
Do iWorkerObj.$init(Params)
Do iWorkerObj.$start() Returns #F
```

The $run() method is analogous to $start() but provided for debugging and testing purposes only. In this mode, the benefit of the worker object is negated owing to the fact that the worker will run on the same thread as the caller, thus blocking the caller thread until the worker task is complete.

If an error occurs during $init(), $start() or $run(), an error message is returned via the object's $errorcode and $errortext properties.

Processing Worker Results

When complete, the worker task causes the main thread to jump into one of the worker object's *callback* methods:

- $completed()
  This method is called with a row parameter defined with two columns: Results: a single-column list containing zero or more SQL result sets (lists). Errors: a two-column list containing ErrorCode and ErrorMsg values.
- $cancelled()
  This method is called (without parameters)if the user calls $cancel() on the worker object whilst it is running. When cancelled, any pending results are discarded.

A library may contain multiple worker objects of a given type. Each may be assigned a separate unit of work (SQL query) and each may be started asynchronously. It is the responsibility of the completion method in each worker object to process its result information and make this available to the main application as/when it becomes available. For example, here is a $completed() method:

```
Calculate List as pRow.Results.1.1 ;;extract the first result set
Calculate Errors as pRow.Errors ;;extract list containing error info
If Errors.$linecount()>0
   Do method reportError(Errors)
Else
   Do method updateWindow(List) ;; see comment below
End If
Calculate iThreadCount as $cinst.$threadCount ;;shows how many
   threads are running
```

If the results returned by $completed() are to be displayed in a window or a remote form in the JavaScript Client, you will have to explicitly redraw the window instance, or in the case of a web form the remote client must contact the server to get updated automatically.

Worker State

The current state of a SQL Worker object may be interrogated by inspecting the object's $state property. This will return either:

- kWorkerStateCancelled – The worker has been cancelled (ready for $init).
- kWorkerStateClear – The worker is in a pre-initialised state (ready for $init).
- kWorkerStateInit – The worker has been initialised (ready for $start).
- kWorkerStateComplete – The worker has completed (ready for $init).
- kWorkerStateRunning – The worker is currently running.

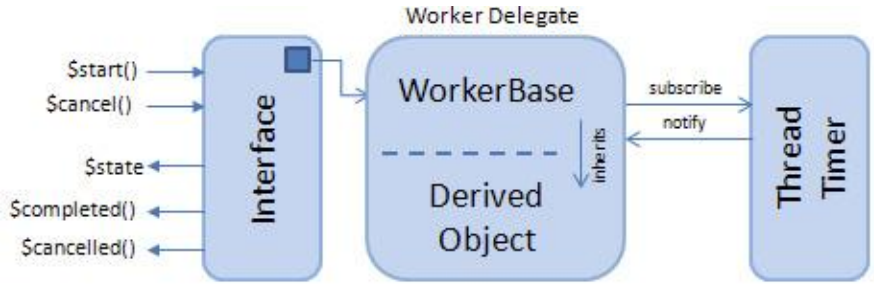
For example:

```
If iWorkerObj.$state=kWorkerStateRunning & iWorkerObj
   .$waitforcomplete=kTrue
   Calculate iMesg as 'Still running (waiting for completion)'
   Quit method
End If
```

How SQL Worker Objects work

A worker object may be thought of as three sub-objects:

- **Interface Object**
  This takes the form of a standard Omnis non-visual object and provides the methods and properties described above
- **Worker Delegate Object**
  Normally created/executed on a background thread, the Worker Delegate performs the actual work of the worker object, calling back to the Interface Object upon completion.
- **Thread Timer Object**
  Declared statically, each Worker Delegate "subscribes" to the Thread Timer object when it starts. Thread Timer polls all worker threads of a given type, prompting them to call back to their corresponding Interface Objects upon completion.

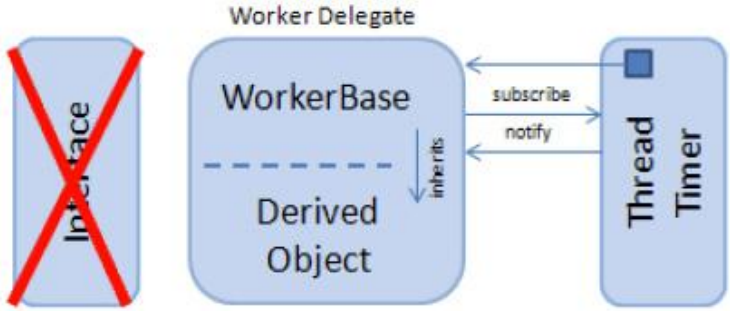


Behind each Worker Object, there are hidden Worker Delegate and Thread Timer objects.

Detaching Worker Processes

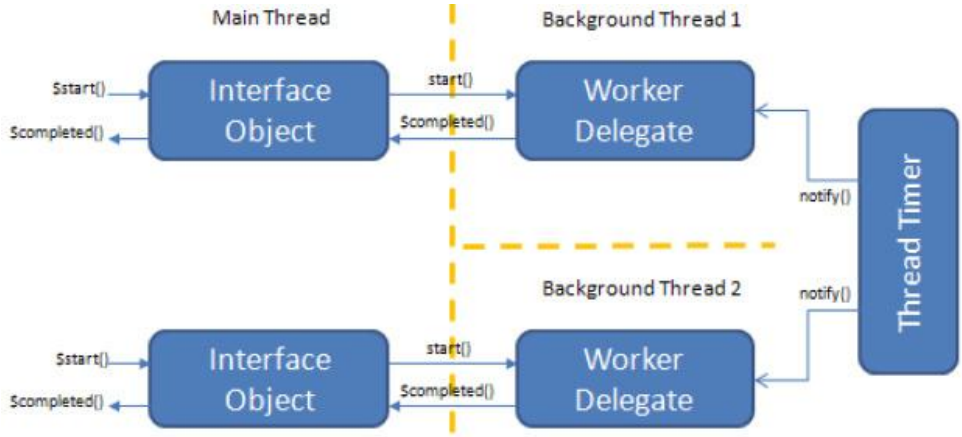
When instructed to $start(), the Worker Delegate completes its work before calling back to the Interface Object's $completed() or $cancelled() method. For $run(), $completed() is *always* called since the worker object blocks- preventing cancellation.

For $start() however, the Interface Object may legitimately go out-of-scope or otherwise get destructed before the worker thread completes. In this situation, the worker thread has no object and hence no $completed()/$cancelled() method to call back to. Any results or error information will therefore be discarded.



A detached Worker Delegate. The Interface Object may have gone out-of-scope or may now point to a new delegate.

Upon destruction of an Interface Object, the object transfers ownership of the worker delegate to the Thread Timer object. One Thread Timer object monitors all Worker objects of a given type and deletes any orphaned worker processes it owns upon completion.



A single Thread Timer object monitors all Worker Delegates of a given type. An Interface Object can initiate and receive results from a single Worker Delegate but there can be multiple Interface Objects.

Discarding Running Processes

In the case where the Interface Object remains in scope, it is possible to call $init() and $start() whilst the worker object is still running a previous task. In this case, the $waitforcomplete property determines whether the running process should be allowed to run to completion and call back to the Interface Object to signal completion.

If $waitforcomplete is kFalse, the running process is detached from the Interface Object as if the Interface Object were about to go out-of-scope. In this case however, a new Worker Delegate object is created which is then used to execute the new worker process and potentially call back to the Interface Object when complete.

If $waitforcomplete is kTrue, Worker Main returns an error to the Interface Object if an attempt is made to re-use the worker object while the Worker Delegate is still running. In this case, the worker object cannot be re-used until $completed()/$cancelled() has been called and the $state changes to indicate completion.

A worker object with its $waitforcomplete property set to kFalse, effectively becomes a "fire and forget" worker object, for example allowing a succession of INSERT, UPDATE or DELETE statements to be continuously dispatched to separate threads using the same worker object.

Cancelling Detached Processes

By default, orphaned worker threads are allowed to run to completion. When a worker process becomes orphaned it may be preferable to issue a cancel request to the worker, especially where it may be processing a SELECT statement- for which the results will not be retrievable once detached from the Interface Object. This is achieved by setting $cancelifrunning to kTrue before the worker object gets re-used or destructed.

If $cancelifrunning is set to kFalse (the default), orphaned worker threads run to completion before discarding their results and being destructed by the Thread Timer object.

Component Icons

You can now use images that are 1.5 and 2 times the resolution (pixel density) of images that are used for standard monitors, which means such images will be rendered in high definition (HD) on the most recent smartphones and tablets, such as the Apple iPhone 5 or Samsung Galaxy S4. To support the use of HD images, there is a new mechanism for storing icon images used for the JavaScript controls in your web and mobile applications.

Note to Existing Users

The existing method of storing icons in #ICONS or an Icon data file and assigning the numeric Icon ID ($iconid) to controls will continue to work, but new JavaScript remote forms do not have the $iconpages property. To support high definition images for controls you can now create and store icon images separately, but inside the Omnis folder and assign them to controls by specifying an icon ID as before. These icon images are stored in the 'html/icons' folder in your Omnis development folder and in the Omnis App Server when you deploy your application.

When Omnis references an Icon ID it will search for the icon in the 'html/icons' folder, then it will search the #ICONS in the current library and then any other Icon datafiles. The new icons will only be available to JavaScript Client remote forms (and for window classes in the fat client). The Web Client plug-in and the iOS Client will not be able to use the new icons, since those clients only support the built-in icon page system.

Creating Icon Images

The new icon support means that you can create your icons in any third-party image editing software and place the images directly in the Omnis tree without having to import them into Omnis using the Icon Editor.

The icon or image files must be saved using the PNG file type and placed in a sub-folder of the 'html/icons' folder in the main Omnis product tree (note this is not the same icons folder in the root of the Omnis tree which contains the built-in icon data files). Each sub-folder represents what is called an **icon set** which is a named collection of icons. The name of the sub-folder in the icons folder becomes the name of the icon set which will then appear in the icon selection dialog. Note that an icon set cannot be named 'datafile' or 'lib' since those names are already used in the icons folder. Any files and folders that do not conform to the naming conventions are ignored.

Image File names

Each image file within an icon set must conform to the following naming convention:

`<text>_<id>_<size><state>_<resolution>.png`

- <text> is the name of image. This string is used in the icon picker dialog when you set an object's $iconid in the Property Manager.
- <id> is the positive integer id to be used as the icon id. It can be in the range 1 to 10000000.

- <size> is the CSS pixel size of the image, i.e. the resolution independent size of the image, meaning that for all resolutions of the same image this has the same value.

The value of <size> has the form <width>x<height>, where the values 16x16, 32x32 and 48x48 are special values since they correspond to the standard icon sizes supported by Omnis.

- <state> is the checked, highlighted, or normal state of the icon for multi-state icons and can be one of the following:
  an empty string for the normal state of the icon
  "c" is the checked state of the icon
  "h" is the highlighted state of the icon
  "x" is the checked highlighted state of the icon
- <resolution> is the factor by which the pixel density is greater than a standard monitor and is one of the following:
  "_2x" for HD devices such as the Retina display
  "_15x" for some devices e.g. certain Android phones that have a 1.5x pixel density.
  an empty string is the default and is for standard resolution devices, equivalent to _1x

Example file names are:

pencil_1657_16x16.png	Normal state 16x16 icon with ID 1657 for standard resolution devices
pencil_1657_16x16_2x.png	Normal state 16x16 icon with ID 1657 for HD resolution devices
check_1658_32x32c_2x.png	Checked state 32x32 icon with ID 1658 for HD resolution devices

Note that the image file names are case insensitive and they must be unique across all platforms and file systems (that is the case of file names is ignored).

You do not have to create an icon image for all resolutions – Omnis will use an icon image closest to the resolution being referenced, scaling as appropriate, and as with all image scaling it is better to force Omnis to scale an image down than to scale it up. Therefore, you may like to provide the highest possible resolution image for your icons and allow Omnis to scale the images to display the lower resolutions.

When the JavaScript Client connects, it sends its resolution to the Omnis App Server. This allows the server to use the appropriate icon when setting iconid properties in server methods.

Non-standard Size Images

You can create images with a size other than the standard sizes (16x16, 32x32, 48x48) by creating the image at a non-standard size and including the image size in the file name when the file is saved. For example, you can create an image 100x200 pixels and name it something like "mygraphic_1688_100x200.png". Existing users should note that this is the equivalent of an 'Icon Page' in the existing icon support.

Setting the Icon set in your library

Libraries have a new preference $iconset. This is the name of the icon set to be used when resolving icon ids for the JavaScript client and the fat client in the current library. When using this library, and when looking up an icon for the fat client or JavaScript client, Omnis will search for icons within this icon set before following the current icon search path for the library. In this case icons present in the icon set will take precedence over those in #ICONS, omnispic.df1, etc.

Setting the Icon ID for objects

When you set the $iconid of an object using the Property Manager, the icon set for the current library will be shown in the Icon picker dialog ($iconset must be set for the library for the icon set to appear) allowing you to select one of the icons in the set. You can select the icon required and the Icon ID will be assigned to $iconid for the object.

Errors

Any errors created while setting the icon ID for objects are sent to a file called iconsetlog.txt located in the html folder.

Assigning a URL for images

When you set the $iconid of a JavaScript client object you can also assign a URL. In server methods, if the value being assigned is a character value that contains a "/" character then Omnis treats it as a URL generated by the iconurl function (meaning that it can contain alternative icon files for the different client resolutions, and also that the server will pick the correct icon for the client resolution).

In client methods, if the value being assigned is not an Icon ID (a literal integer or integer + icon size constant) then Omnis treats the value as a URL generated by the iconurl function on the server, and the client picks the correct icon for its resolution.

You could generate the required URLs with iconurl() (see below) in the $construct() method of your remote form, and store them in an instance variable list which could then be used in client executed code to assign the correct image to each object.

Image handling for tree lists

For the JavaScript Tree control, the iconid column is now an iconurl column, and the $iconurlprefix property is now redundant although existing libraries that use $iconurlprefix will continue to work. Instead, the iconurl column should be defined to be of type character, and it should be populated using a new server-only function, iconurl(iconid), which returns a URL string containing the name of the image file or a semi-colon separated list of file names if an icon exists in more than one resolution. This enables the client to pick the correct icon for its resolution.

Deploying HD Icons

You need to copy your icon sets and images files to the Omnis App Server when you want to deploy your web or mobile app. If the icons are not copied to the Server tree they will appear to be missing from your app.

Standalone Client Apps

Note that for standalone apps the icons needed for your mobile app will be bundled in the SCAF. If any icons change on the Omnis App Server they will be updated on the client when the standalone application files are updated.

Exporting Icons from an Icon Datafile

You may want to use some existing icons located in an Icon Datafile as separate files and either add to or replace some of them with higher resolution versions. To enable you to export existing icons there is a new tool in the Tools>>Add Ons menu, called the 'JS Icon Export' tool, which is available in the 'Web Client Tools' dialog (scroll to the bottom of the list of Web Client tools). The 'JS Icon Export' tool will export all the icons in a selected Icon Datafile and place them in a folder in the 'html/icons' folder, applying the correct image file names. The $iconid property of a control will now reference the external image file and not the icon datafile image.

PDF Printing

There is a new printing device (external component) to allow you to print a report from Omnis Studio to a PDF file and display it in the JavaScript Client in the end-user's web or mobile browser. There are two new client commands (used with $clientcommand) to allow you handle PDF reports in the JavaScript Client (if the end user's device supports PDF).

The PDF Device is available for Windows and OS X. Various supporting files are located in the 'python' folder within the main Omnis Studio folder, and the PDF component itself is in the 'xcomp' folder: these files need to be present in the Omnis App Server when you deploy your web or mobile app.

Note for existing users

The new PDF device replaces the "Custom Report Device" available in previous versions (for Linux only). The constant kDevOmnisPDF is now used for the new PDF device, therefore if you have used the "Custom Report Device" on Linux in previous versions, you must now use kDevOmnisPrintPDF on Linux to refer to the old print device. In addition, the device parameters for the Linux PDF device have been renamed to kDevOmnisPrintPDFAutoOut and kDevOmnisPrintPDFAutoCommand.

Fonts

The PDF device will only work with Reports that use TrueType fonts, and using other fonts may cause an error during PDF generation. Specifically only fonts contained in ".ttf", ".ttc" or ".dfont" files can be used. Therefore you must choose TrueType fonts for your report if you intend to print using the new PDF device.

Ensure that all fonts used in your PDF reports are in the specified locations within the rl_config.py file stated under the 'T1SearchPath' section within the rl_config.py file which can be found in the following folder:

```
omnis\python\App\Lib\site-packages\reportlab
```

Any installed font that is used in your report and not in one of these locations will result in an error message when attempting to print to the PDF Device.

PDF Print Destination

The new component is called OmnisPDF and appears as a new print option in the Print Destination dialog, available to end users from the main File menu. If the end user selects PDF as the report destination, then when they print a report, it will either be sent to a report file specified in $prefs.$reportfile, or if this preference is empty Omnis will prompt the user to select the path of the output PDF file. Note that this mechanism can be overridden programmatically by using the $settemp method (see below).

Printing PDF Using Code

To send a report to PDF programmatically, you can use the following code:

```
Calculate $cdevice as kDevOmnisPDF
Set report name MyReportClass
Print report
; or
Calculate $cdevice as kDevOmnisPDF
Set reference lReportInst to $clib.$reports.New Report.$open('*')
Do lReportInst.$printrecord()
Do lReportInst.$endprint()
```

In both of these cases the report will be sent to a report file specified in the $prefs.$reportfile preference. Note that there is a separate value of $prefs.$reportfile for each Omnis App Server stack (thread) and the main Omnis thread. If $prefs.$reportfile is empty, and the code is running in the main Omnis thread, Omnis will prompt the user for the destination PDF file; otherwise, if $prefs.$reportfile is empty and the code is running in another thread, Omnis will generate a runtime error.

PDF Device Functions

The PDF device has a number of functions to allow you to set up reports sent to temporary PDF files, to set up document properties, and to add security features such as passwords and encryption.

$settemp()

The $settemp function allows you to specify that the next report will print to a temporary PDF file in the omnispdf/temp folder. The function returns the name of the file that will be created in omnispdf/temp (or an empty string if bTemp is kFalse). You can specify a timeout in minutes whereupon the temporary PDF file will be deleted.

```
Do Omnis PDF Device.$settemp(bTemp,iTimeout) Returns cID
```

bTemp	A Boolean: kFalse means a temporary file will not be used. kTrue means the next report sent to PDF by the current task instance will be written to a temporary file in the folder "omnispdf/temp" in the data part of the Omnis Studio tree. Note that after the next report has been sent to PDF, the stored value of bTemp will revert to kFalse.
iTimeout	An integer: Only used when bTemp is kTrue. If omitted defaults to 10. The time in minutes for which the next PDF file generated by the current task will remain on disk. When the time expires, the Omnis PDF device automatically deletes the file. Note that Omnis automatically deletes any files left behind in omnispdf/temp when it starts up.

Each task instance (including remote tasks) stores its own information set up using $settemp. This allows $settemp to be used in one thread on the Omnis Server without affecting other threads/clients.

$setdocinfo()

The $setdocinfo function lets you specify the author, title and subject properties for the PDF documents generated by the current task. The author, title and subject parameters are all strings, and the function returns kTrue for success.

```
Do Omnis PDF Device.$setdocinfo(cAuthor,cTitle,cSubject) Returns bOK
```

$encrypt()

The $encrypt function sets encryption (security) properties for the PDF documents generated by the current task, and the function returns kTrue for success. The full syntax is:

```
Do Omnis PDF Device.$encrypt(
   cUserPassword
   [,cOwnerPassword='',
   bCanPrint=kTrue,
   bCanModify=kFalse,
   bCanCopy=kTrue,
   bCanAnnotate=kFalse]) Returns bOK
```

The parameters are as follows:

cUserPassword	A character string: The user password for the document. If this is set to empty then none of the other arguments apply and the document will not be encrypted; otherwise the document will be encrypted and the user password and other properties specified by this function will be applied to it.
cOwnerPassword	A character string: The owner password for the document. The default is no password is assigned.
bCanPrint	A Boolean: Specifies if the user can print the PDF document. The default is kTrue.
bCanModify	A Boolean: Specifies if the user can modify the PDF document. The default is kFalse.
bCanCopy	A Boolean: Specifies if the user can copy from the PDF document. The default is kTrue.
bCanAnnotate	A Boolean: Specifies if the user can annotate the PDF document. The default is kFalse.

Security in Third-party PDF readers

The optional security parameters will be applied to the PDF file if you include them in the $encrypt() function, but you should note that the third-party PDF viewer the end user is using may not support these settings or may just choose to ignore them. The password specified in cUserPassword should be interpreted by all PDF readers.

PDF Printing in the JavaScript Client

PDFs generated by the new device can be used with the JavaScript Client. There are two new client commands (used with $clientcommand) called "showpdf" and "assignpdf" to display PDF files on the client. You should avoid generating large PDF documents to use with the JavaScript Client since the generated PDFs are streamed from the Omnis App Server (i.e. via a Web Server if one is being used). An alternative would be to output the PDF document into the Web Server's file system, and use a URL to the PDF on the Web Server to send it to the client.

showpdf

The **showpdf** client command opens the specified PDF in a new window or tab. There is no control over whether the PDF is opened in a new window or tab: typically this depends on the end-user browser settings, including their setting for popups.

```
Do $cinst.$clientcommand("showpdf",row-variable)
```

Where *row-variable* is row(**pdf-id, timeout, pdf-filename**)

- **pdf-id**
  A character string. Either a full pathname of a PDF file on the Omnis server, or an id returned by $settemp

- **timeout**
  An integer being the time in seconds that the client is prepared to wait for PDF generation to finish. Defaults to 60. If PDF generation does not complete in time, or an error occurs, Omnis returns a PDF document containing a suitable error message.
- **pdf-filename**
  the file name of the PDF file

The following method generates a PDF and displays it on the client:

```
Calculate $cdevice as kDevOmnisPDF
Do Omnis PDF Device.$settemp(kTrue,1) Returns lID
Set report name New Report
Do Omnis PDF Device.$encrypt('bob','owner',1,0,0,0)
Do Omnis PDF Device.$setdocinfo('Bob','Title','Subject')
Print report
Do $cinst.$clientcommand("showpdf",row(lID,20))
```

assignpdf

The **assignpdf** client command opens the specified PDF in a PDF viewer control in the current remote form instance. The PDF must be assigned to an HTML control in the remote form which tries to open the PDF file using the PDF viewer installed in the end user's browser.

```
Do $cinst.$clientcommand("assignpdf",row-variable)
```

Where *row-variable* is row(**html-object-name, pdf-parameters, pdf-id, timeout, pdf-filename**)

- **html-object-name**
  The name of an HTML Object control in the current remote form instance. The HTML content of this object will be replaced with that necessary to display the PDF document in a PDF viewer control.
- **pdf-parameters**
  PDF viewer parameters. These apply when the PDF is viewed in a browser that uses the standard Adobe PDF viewer control. They control the look and behavior of the PDF viewer. See the Adobe website for details about the PDF Open Parameters.
- **pdf-id**
  A character string. Either a full pathname of a PDF file on the Omnis server, or an id returned by $settemp
- **timeout**
  An integer being the time in seconds that the client is prepared to wait for PDF generation to finish. Defaults to 60. If PDF generation does not complete in time, or an error occurs, Omnis returns a PDF document containing a suitable error message.
- **pdf-filename**
  the file name of the PDF file

The following method creates a report and assigns it to an HTML control in the remote form.

```
Calculate $cdevice as kDevOmnisPDF
Do Omnis PDF Device.$settemp(kTrue,1) Returns lID
Set report name New Report
Do Omnis PDF Device.$encrypt('bob','owner',1,0,0,0)
Do Omnis PDF Device.$setdocinfo('Bob Smith','Title','Subject')
Print report
Do $cinst.$clientcommand(
        "assignpdf",row("htm","toolbar=1&zoom=20",lID,10))
```

If you set the $html property of the HTML Object control to "<div %e> " the PDF viewer will have the same border, position and dimensions as the designed control on the remote form. Note that this command will not work on Android with the default web browser, since it does not support the application/pdf plug-in. If the application/pdf plugin is not available on Android, it executes showpdf instead.

Print PDF Example

The following methods will allow the end user to print a report to a PDF file: these methods use the built-in PDF Device methods and the client commands.

```
; button to open PDF report
; create var lID (Char)
On evClick
   Do $cinst.$createReport() Returns lID
   Do $cinst.$clientcommand(
        "showpdf",row(lID,60))
; or button to display PDF report in current remote form
; create var lID (Char), oHTML is an HTML obj on the form
On evClick
   Do $cinst.$createReport() Returns lID
   Do $cinst.$clientcommand(
        "assignpdf",row("oHTML","toolbar=1&zoom=20",lID,20))
```

Here is the code for the $createReport() remote form method:

```
Calculate $cdevice as kDevOmnisPDF
If iSaveCopy  ;; linked to check box on the form
   Do Omnis PDF Device.$settemp(kFalse) Returns lID
   Calculate lSaveLocation as left(sys(10),rpos(sys(9),sys(10)))
   Calculate lSaveLocation as
        con(lSaveLocation,"savedReports",sys(9),iFileName)
   Calculate $prefs.$reportfile as lSaveLocation
Else
   Do Omnis PDF Device.$settemp(kTrue,1) Returns lID
```

```
End If

Set report name repNice
If iEncrypt
   Do Omnis PDF Device.$encrypt(
        iUserPass,iOwnerPass,iCanPrint,iCanModify,
        iCanCopy,iCanAnnotate) Returns #F
End If
Do Omnis PDF Device.$setdocinfo(iAuthor,iTitle,iSubject) Returns #F
Print report

If iSaveCopy
   Quit method lSaveLocation
Else
   Quit method lID
End If
```

Localization for the JavaScript Client

There is a new mechanism for managing localization for web and mobile apps that use the JavaScript Client. You can now use a new Tab Separated Value (TSV) format to store tables containing alternative strings to support multiple languages in your remote forms.

Note that for compatibility the old way of storing string tables internally (.stb format) and handling them via $stringtabledata and $stringtabledesignform still works with Omnis Studio 6.0.

String Table Format

The string table external component in Studio 6.0 now supports two disk formats for storing string tables. These are the existing .stb format and a new tab separated value (TSV) format, which has the following features:

- Data is stored in a UTF-8 encoded text file, as a series of rows of fields.
- Fields in the data are separated by tabs.
- Each field is enclosed in double quotes.
- Double quotes in field values are escaped as a pair of double quotes.
- Field can contain newline characters.
- Each row is separated by a newline outside the contents of a field.
- The fields in row 1 are the column names for the string table.

The external component selects the new TSV format automatically if the string table file has the extension ".tsv". The current String Table editor supports the new TSV format. The new string table file should be stored in the same folder as the library file.

If you open the Catalog while editing a remote form or remote task that has a string table associated with it (via $stringtable), then the Catalog automatically loads the $stringtable if it is not already loaded, or reloads it if it has changed on disk.

Localizing Remote Forms

Remote tasks have a new property, $stringtable which is the name of the string table for the current library and shared by all JavaScript client remote form instances in the remote task. Omnis automatically loads it and names it using the library name and file name.

When a client connects and the remote task is using the new $stringtable property, Omnis loads this string table if it is not already loaded; if the modify date of the table has changed, Omnis reloads it from disk. If the client does not have an up to date copy of the string table, Omnis adds the string table to the connect response returned to the client. The client then caches the table so that it will only be sent to the client again if it changes (the cache entry associates the table with the library and remote form name). The client then uses the table for the stgettext() function, and properties assigned with the $st prefix, as in the previous version. Omnis unloads the string table automatically when the library closes.

In addition, server methods can also use stgettext() to look up strings for the client locale (either the locale received from the client, or the locale set using $stringtablelocale). If the locale is not present in the string table, stgettext() will return values from column 2 of the string table.

The server only loads a single copy of the string table for all client instances created by the library. Note that if you change the string table while a client is connected, and a new client connects, server methods will start using values from the updated string table, whereas the initial connected client will continue to use the old string table.

Standalone Client

If you run your application in the Standalone client you have to set the new library preference $serverlessclientstringtable to specify which string table to use for the remote form instances in the SCAF for the library. The property takes the name of a string table which is a tab-separated value .tsv file in the same folder as the library.

Localizing Error Strings

There are a number of strings that can appear in error message and other dialogs on the JavaScript Client. These are built into the client and are all in English by default, but for Studio 6.0, you can provide your own translations for some or all of these strings. Note that some of the error messages are very unlikely to appear in the final deployed version of your app, especially if you have thoroughly tested your app and eliminated all the errors, so it not entirely necessary to translate most of these error strings.

To override any of the default strings, you can add your own translated string to the HTML page containing the JavaScript Client. Each of the strings in the JavaScript Client has an object name so you can reference by name, adding your own text for the sting separated by a colon. For example, the following would override the default message string for the Omnis App Server timeout (which occurs as specified in the $timeout property):

```
<script type="text/javascript">
   jOmnisStrings.en = { "comms_timeout":"A timeout has occurred" };
</script>
```

The following strings are present in the JavaScript Client, where \x01 is a place-holder which is replaced by parameters added to the string when it is called by the client; you should retain \x01 in your translated text.

Error string object	Error 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_tree_invmode	Invalid data mode for tree
error	Error
omn_form_ctrlinst	Failed to install the control \x01. Possible missing class script
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_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_sfsnotthere	\x01: A subform set with name \\x01\ does not exist
omn_inst_xmlhttp	Failed to initialize XMLHttpRequest
omnis_badhtmlesc	Invalid HTML escape
omnis_badstyleesc	Invalid style escape sequence
omnis_convbad	Error setting \x01: variable type \x01 not supported by JavaScript Client

Error string object	Error string text (English default)
omnis_convbool	Error setting \x01: data cannot be converted to Boolean
omnis_convchar	Error setting \x01: data cannot be converted to Character
omnis_convdate	Error setting \x01: data cannot be converted to Date
omnis_convint	Error setting \x01: data cannot be converted to Integer
omnis_convlist	Error setting \x01: data cannot be converted to List
omnis_convlistrow	Error setting \x01: cannot convert List to Row
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

Rich Text Editor Control

There is a new JavaScript control that allows the text in a field to be edited by the end user – this is very useful for applications in which you want to allow end users to edit content and will be suitable for many types of web and mobile apps.

The JavaScript **Rich Text Editor** Control (jsrich) is available in the JavaScript Component Store and can be used instead of a regular edit or multi-line edit field. The text data for the control is stored in the instance variable assigned to $dataname. You can allow text editing in the control by setting $showcontrols to kTrue where upon the text content in the field will become editable: on mobile devices this places the cursor in the field and opens the soft keypad ready for typing. The text editing controls in the field will appear at the top of the control and will allow the end user to format the text, including bold, italic, underline, and so on. The text data in the control is HTML markup and can include formatted text such as ordered (numbered) and unordered lists (bullets). End users can also insert images using the Paste option.

Properties

Property	Description
$dataname	The name of an instance variable to store the text, or a column in an instance row variable
$showcontrols	Set to kTrue to show the Rich text editing toolbar
Standard	$align $alpha $autoscroll $backalpha $backcolor $bordercolor $componentctrl $componentlib $contextmenu $disablesystemfocus $edgefloat $effect $enabled $events $fieldstyle $font $fontsize $fontstyle $height $horzscroll $ident $jscustomformat $jsdisplayformat $left $linestyle $name $objtype $order $screensizefloat $textcolor $tooltip $top $userinfo $vertscroll $visible $width

Localizing the Rich Text Editor

There are various strings in the Rich Text Edit control that can be localized in the string table for the remote form containing the control. You must use the following string table ids to replace the default text for the controls in the text editor.

Tooltips for buttons

rt_bold	rt_subscript	rt_italic
rt_superscript	rt_underline	rt_strikethrough
rt_justifyleft	rt_removeformat	rt_justifycenter
rt_indent	rt_justifyright	rt_outdent
rt_justifyfull	rt_textcolor	rt_insertorderedlist
rt_backgroundcolor	rt_insertunorderedlist	

Text displayed on controls

rt_fontsize and rt_fontfamily.

Dynamic Tree Lists

The content inside a Tree List Control can now be built dynamically as the end user expands a node. In previous versions, the entire contents of the tree list had to be built and sent to the client, including the contents for all unexpanded nodes, which for large lists created quite an overhead. Building the tree list data and displaying content in a tree list can now be better optimized with the ability to build node contents "on the fly" when required.

Creating Dynamic Trees

To use a Tree Control in dynamic mode you need to set its $datamode property to kJSTreeDynamicLoad. Note that dynamic mode can only be used for 'single-select' trees, and attempting to assign kJSTreeDynamicLoad to $datamode will fail if $multipleselect or $checkbox for the tree control is kTrue.

As in previous versions, a dynamic tree still requires a list identified by $dataname, but the list need only contain the initial content of the tree, that is, the content for the root or parent nodes. After the list content is changed, the tree reloads its content from the list.

Dynamic trees also use lists to set the content of expanded nodes and to add nodes programmatically to the tree. The lists all have the same structure: each list represents an ordered set of nodes with the same parent (or no parent in the case of the $dataname list), and the columns are as follows:

- Column 1: Text. The text displayed for the node.
- Column 2: Icon. Only used when $showicons is set to kTrue. This is a line number in the list identified by $nodeiconlist, or zero if the node does not have an icon. $nodeiconlist is described below.

- Column 3: Ident. A unique positive integer that identifies the node. Cannot be the same as the ident of any other node in the tree. The tree control throws an exception (resulting in a message box displayed on the client) if you try to use a duplicate ident.
- Column 4: Tag. A string associated with the node. Any value that is useful to the developer. Need not be unique.
- Column 5: Tooltip. The tooltip string for the node, displayed when the user hovers the mouse over the node. Leave empty for no tooltip, although on some browsers nodes may inherit their tooltip from their parent node if the tooltip is empty.
- Column 6: Text color. The text color for the node (an integer RGB value). Zero means use the $textcolor of the tree.
- Column 7: Flags. Integer flags. A sum of zero or more of the following constants:
  - kJSTreeFlagEnterable. The node text can be edited (this works with the existing evRenamed event).
  - kJSTreeFlagHasChildren. The node has children.
  - kJSTreeFlagExpanded. The node will be immediately expanded. If you set this flag you must supply the child nodes using a node list supplied as the children column.
  - kJSTreeFlagDiscardOnCollapse. If set, when the user collapses the node, the tree deletes the node contents. This means that the next time the user expands the node, the tree will generate an evLoadNode event; evLoadNode is described later in this document.
- Column 8: Children. If the kJSTreeFlagHasChildren is present, you can pre-populate the node content by using a nested list to specify the children. If you do not supply any children using this list, then you can supply them later by using the evLoadNode event (see below). The content of this column is ignored if the kJSTreeFlagHasChildren is not present. You can nest children lists arbitrarily deep (within reason).

$nodeiconlist allows you to specify the icons to be used with the tree. These must be available when the tree is updated from the dataname list. $nodeiconlist must be the name of an instance variable list with at least one column which must be a character column containing icon URLs. You can populate each URL in the node icon list using the iconurl() function, for example:

```
Do iNodeIcons.$define(iNodeIcon)
Do iNodeIcons.$add(iconurl(1710))
Do iNodeIcons.$add(iconurl(1711))
Do iNodeIcons.$add(iconurl(1712))
```

Populating Expanded Nodes

Dynamic trees have a new event, evLoadNode, which allows you to populate the tree on demand, by only populating node content when a node is expanded. When using the kJSTreeDynamicLoad mode for $datamode, evLoadNode is generated so that you can set

the content of the node by setting a new property $nodedata which is the name of a list containing the expanded node content. For other settings of $datamode, evLoadNode is generated when the user expands a tree node.

The evLoadNode event has two parameters, pNodeIdent and pNodeTag, corresponding to the node that is being expanded. In the event processing for evLoadNode, you can set a new tree property, $nodedata, to a node list representing the content of the node (with the above column format); if the event processing fails to set this property, the tree control sets the node content to empty.

The list assigned to $nodedata can specify nested children if desired.

$nodedata is a runtime-only property that can only be set.

Manipulating Tree Nodes

Dynamic trees support 'node actions' to allow you to manipulate the nodes in a Tree List programmatically. For example, you can expand or collapse a node, and you can add, delete or rename modes. You execute a node action by assigning a row variable to the $nodeaction property of the tree. The supported actions are as follows:

- **kJSTreeActionExpand**
  row(kJSTreeActionExpand, ident). Expands the node with the specified ident if it is not already expanded.
- **kJSTreeActionCollapse**
  row(kJSTreeActionCollapse, ident). Collapses the node with the specified ident if it is not already collapsed.
- **kJSTreeActionRename**
  row(kJSTreeActionRename, ident, newname). Renames the node with the specified ident to the new name.
- **kJSTreeActionDelete**
  row(kJSTreeActionDelete, ident). Deletes the node with the specified ident (also recursively deletes node children). If the current node is deleted, an evClick event will be generated to inform the application of the new current node.
- **kJSTreeActionAdd**
  row(kJSTreeActionAdd, ident, position, nodelist). Adds the nodes in the nodelist to the tree, where ident specifies the parent node of the nodes in nodelist; to add a new root nodes specify ident as zero. The position parameter can be one of:
  - -1 to add the new nodes before any existing children of the node specified by ident
  - 0 to add the new nodes after any existing children of the node specified by ident
  - a child node ident. New nodes are added after the child node with this ident. If no such node exists, the new nodes are added after any existing children.

$nodeaction is a runtime-only property that can only be set.

Collapsing a Node

The tree control now generates evCollapseNode when the user collapses a node. This applies to all trees, not just dynamic trees.

The Current Node

The current (selected) node in the tree is no longer represented by a list line. Instead, there is a new property, $currentnodeident that applies to dynamic trees. When the user changes $currentnodeident (by clicking on a node), the control generates evClick. In addition, the developer can assign $currentnodeident (and read its value in client-executed methods). $currentnodeident is a runtime-only property.

Linked Lists

A number of enhancements have been made to the Edit and List controls to allow you to create a "Linked List" that can be updated as the end user types into your remote form. The combination of an Edit control with key presses enabled and a List control using some new properties will allow you to create a new type of dynamic list that has the ability to update in response to what the end user types into the edit box. The new Linked List is in effect like a Combo box, but with the extra ability to update itself as the user types. To enable this feature, edit controls can now detect certain key presses and can be linked to a list control, while for lists themselves there is a new property to display selected lines only.

Detecting Key Presses

Edit controls now have a new event, evKeyPress. It has a single event parameter, pKeyList, which is a three column list containing the keys entered since the last evKeyPress event (or since typing started) with a row for each key in the order the keys were pressed. The new Key press detection was added to support Linked Lists, but it could be used by itself for other purposes.

Each key in the list is either a data character or one of a limited set of system keys (note that not all keyboard keys are supported, such as function keys are not supported). Transitions in shift state, ctrl state, and so on, do not result in a key in the list, so if the user types Shift+a, the character in the list will be A. The supported system keys are:

- Backspace
- Tab
- Escape
- Insert and Delete
- Up, Down, Left, and Right arrow key
- Page Up and Page Down
- Home and End
- Return

The columns in the list are as follows:

- Colum 1: If the key is a system key, column 1 is the keyboard constant value representing the key, such as kEscape. Otherwise column 1 is #NULL.
- Colum 2: If the key is a data key, column 2 is the character data for the key. Otherwise column 1 is #NULL.
- Colum 3: Is the sum of the following keyboard modifier constants, representing the state of these modifiers at the point the key was added to the list: kJSModShift, kJSModCtrl, kJSModAltOrOption, kJSModCmd.

In addition, there are some properties which control when evKeyPress events are generated. The Omnis preference $keyeventdelay is the minimum number of milliseconds (0-2000) between evKeyPress events. The first evKeyPress will also be delayed for this duration. This allows you to throttle keyboard events in the case where they will be executed on the server, and therefore reduce the load on the server. If true, the preference $systemkeys specifies that evKeyPress events include system keys which do not change the value of the data. If false, only system keys such as backspace are included as they potentially change the data.

While the user is typing, the edit control in the user interface remains enabled (unlike normal event processing where the entire user interface is usually disabled). The value of the instance variable associated with the edit control reflects the current content of the entry field when evKeyPress is generated.

Creating Linked Lists

The edit control has a new property, $linkedobject, which is the name of a list control on the current remote form used to display suggestions to the user if the Edit control enables the evKeyPress event. You need to add the code to populate the list in response to evKeyPress, based on the current value of the instance variable specified for the Edit control.

When the end user types into the Edit control, the linked list will appear automatically. It is not necessary to process any events for the list control, since the dynamic list behavior is determined automatically by being linked in this way. Therefore you would expect the following to happen when end users are using a dynamic linked list:

- Clicking on a line in the open list will set the edit control to the value of the clicked line.
- Pressing up or down arrow when the focus is on the list control will set the value of the edit control to the new current line of the list.
- Pressing the down or up arrow when the edit control has the focus and when the list is not visible will open the list, set the edit control value to the selected line in the list, and move focus to the list.
- Typing into the edit control when the list is closed will open the list and keep focus on the edit control.
- All the following will close the list: pressing return when the list has focus; pressing escape, or clicking away from the list or the edit control.

Optimizing the Linked List

There is a new property for List Controls called $selectedlinesonly which specifies that only the selected lines in a list will be displayed (this only applies when $ischecklist is false), which in the context of Linked lists allows you to filter the content of a list that is linked to an Edit control. You should optimize the $event for the edit control in the following way:

- Make $event for the Edit control execute on the client.
- On evKeyPress, if the list of suggestions is empty (or no longer suitable for the data), call a server method to build the list (with lines selected based on the value of the edit control).
- In subsequent evKeyPress events that can use the pre-built list, perform a list $search to change the selected lines to the ones which are suitable for the new value of the edit control.

In this way, the list of suggestions is cached on the client, and updates simply by changing the selected lines with a search in your code.

When $selectedlinesonly is true, the processing involving the usual click events and so on all use the line number of the *data* in the list, not the lines displayed in the list.

Note that if you use the $evenrowcolor property when $selectedlinesonly is true, the even row color applies to the numbers obtained by counting the displayed lines, rather than using the line numbers of the data in the list.

Data Grids

There are a number of enhancements in the JavaScript Data Grid component to increase its customization and richness for presenting data to the end user in a grid layout.

Data Grid Cell Formatting

You can now apply your own formatting to individual cells in a Data Grid. For user-defined Data Grids (where $userdefined is set to true) and where a column has its columnmode set to kJSDataGridModeCustomFormat, you can customize the HTML used to layout or format an individual cell in the grid.

When the grid is rendered (this occurs on demand e.g. when scrolling to make new data visible), it calls the object client method:

```
$formatcell(list line number, designed grid column number)
```

which you implement to return html for the formatted cell contents. This HTML can, within reason, be anything you like: you can also just return a text string. To assist this, there are two new calls you can make:

- styledtohtml(text)
  the styledtohtml(text) function returns the HTML representing the text string containing embedded styles inserted using style() function. This is a built-in function,

under the General tab of the Catalog, which must be run in a client-executed method in the JavaScript Client only.

- $addcolorcss(cClassName,iRgbColor,iAlpha)
  is a method of the data grid object (as such it can only be executed in client-executed methods). Call this in $init to add your own background color class to use with the html returned by kJSDataGridModeCustomFormat.This class takes browser specific issues with transparency into account.

For example, you can use the following method:

```
Do $cinst.$objs.datagrid.$addcolorcss("myclass",rgb(255,0,0),128)
```

in $init, and then do the following in $formatcell:

```
Calculate columnvalue as iList.[row].[col]
if (columnvalue == "bad")
   Quit method con(
      '<div class="myclass"
         style="padding:0;margin:0;height:16px">
         ',columnvalue,' ','</div>')
   Quit method columnvalue
```

Using transparency in the CSS background allows the selection color to show through the formatted cell.

Data Grid Header Formatting

You can create your own formatting for column headers by adding a client-side method called $formatheader which takes two parameters:

- Parameter 1: the text for the column header
- Parameter 2: the design grid column number (1-n)

The return value is HTML to use for the header, for example, for a bold header:

```
<b>Param 1</b>
```

For red text:

```
<span style="color:red">Param 1</span>
```

For right justified text (using float so that sort indicators still appear):

```
<div style="float:right;">Param 1</div>
```

You can reassign the column name to force a call to recalculate the HTML for the column header, even if the text has not changed.

Data Grid Column Data Type Formatting

When you set $columnmode to kJSDataGridModeFormatted, the mode acts like kJSDataGridModeAuto, in that the data grid automatically handles the data based on its type. However, the grid formats the data using some new column properties, $columndateformat, $columndateformatcustom, and $columnnumberformat, rather than the $js...format... properties.

You can use an integer column data type to represent a checkbox. To do this set $columnmode to kJSDataGridModeFormatted and set the $columnnumberformat to "bool". This will cause integer data to be treated as Boolean, where non-zero means true, and zero means false. If the end user updates the grid using the check box, 1 will be stored in the list for true, and zero for false.

Grid and List Scrolling

JavaScript Data Grids now have $vscroll and $hscroll properties which allow you to scroll a grid vertically or horizontally at runtime in the client browser; note these properties are write-only meaning that you cannot return their values at runtime. This feature is also available for simple List components.

The vertical scroll value assigned using $vscroll is the position of the scroll bar according to row number in the control. The horizontal scroll value assigned using $hscroll is the designed grid column number for a data grid, or a pixel offset for a list.

Data Grid Column Data

The JavaScript Data Grid property $columndatacol can now use a column name, rather than a column number.

Integers

64-bit Integers

You can now define numbers as 64-bit Integers. When defining a number (e.g. as a schema column or as a variable in the Method Editor), the numeric types are now split into:

- **Integer**
  Short – for integers in the range 0 to 255, as in previous versions (kShortint)
  32 bit – for storing 32 bit numbers (k32bitint)
  64 bit – for storing 64 bit numbers (k64bitint)
- **Number**
  with various Real number subtypes, as in previous versions.

The range of the new 64-bit Integer is:

-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807, or $-(2^{63})$ to $2^{63} - 1$

Long integer functions

The existing 'binary to long' functions bintolong() and binfromlong() have been renamed to bintoint32() and binfromint32() which convert 32-bit Integers, and there are two new functions bintoint64() and binfromint64() for converting 64-bit Integers. Note these new functions cannot be executed in client methods in the JavaScript Client.

The kLongint constant for data subtypes has become k32bitint, and there is a new subtype k64bitint.

DAMs support for 64-bit Integers

The Studio 6.0 DAMs perform additional data type mappings between Omnis 64-bit integers and the corresponding large integer data type on the database server. For most databases this will be BIGINT. The notable exception is Oracle which uses NUMBER(19,0) instead.

Note also that BIGINT UNSIGNED columns will be converted to signed 64-bit Integers when fetched into Omnis. In order to preserve such values, a *CAST(column as CHAR)* function can be used to fetch the value into a character field.

Where schemas and lists are defined using Integer 64-bit columns, the session object's $coltext() and $createnames() methods now return the appropriate SQL data type. Integer 32-bit columns retain their previous behavior.

There is a new session property to truncate 64-bit integers to 32-bit if required:

- **$fetch64bitints**
  If kTrue (default), 64-bit integers are fetched into 64-bit Integer fields. If kFalse, they are fetched as 32-bit Integers and truncated accordingly. This property provides backward compatibility with the old Web Client plug-in which does not support 64-bit integers.

File Classes

You can use 64 bit integers in file classes, provided that a field (column) is not indexed. In Integer subtype droplist in the file class editor, the 64 bit subtype is available but only when the field is not marked as indexed, otherwise if the field is indexed 64 bit is not present: the 64 bit subtype is not allowed in compound indexes either. The notation to manipulate file classes and indexes does not allow 64 bit integers to be used for indexed fields.

Number Formatting

All JavaScript controls that can display *number data* now have the property $numberformat, which specifies how Number and Integer data is formatted or displayed in the control. The JavaScript controls affected include the Edit Control, Combo box, Data grid, Droplist, Hyperlink list and standard List control. The formatting is used when the control displaying the data does not have the focus, so for example, the formatting is applied when the end user tabs or clicks away from the number field.

Note to existing users: To accommodate the addition of $numberformat, the date formatting properties introduced in Studio 5.2, $jsdisplayformat and $jscustomformat, have been renamed to $dateformat and $dateformatcustom, respectively, but their behavior is unchanged.

The $numberformat property uses a single % format tag for the number followed by one or more elements, for example, the number format %.2F displays a number with 2 decimal places with a thousand separator. The following elements are available (in this order):

An optional "+" sign that forces to precede the result with a plus or minus sign on numeric values. By default, only the "-" sign is used on negative numbers.

An optional padding specifier used for padding (if padding is required). Possible values are 0 or any other character preceded by a '. The default is to pad with spaces.

An optional "-" sign, that causes the string to left-align the result of this placeholder. The default is to right-align the result.

An optional number that says how many characters the result should have. If the value to be returned is shorter than this number, the result will be padded.

An optional precision modifier consisting of a "." (dot) followed by a number, specifies how many digits should be displayed for floating point numbers. When used on a string, it causes the result to be truncated.

A type specifier that can be any of:

%	print a literal "%" character
b	print an integer as a binary number
c	print an integer as the character with that ASCII value
d	print an integer as a signed decimal number
e	print a float as scientific notation
u	print an integer as an unsigned decimal number
f	print a float as is
o	print an integer as an octal number
s	print a string as is
x	print an integer as a hexadecimal number (lower-case)
X	print an integer as a hexadecimal number (upper-case)

Custom CSS Styles

You can now create your own CSS classes or styles and apply them to the objects in your web and mobile apps, allowing you to have more control of the styling, coloring, and overall design of your apps.

CSS classes for Controls

All the JavaScript components have a new property called $cssclassname which allows you to apply a CSS class to the component. You can add the CSS classes to a file called 'user.css' which is located in the 'html/css' folder in the main Omnis Studio folder. A style can be applied to a control by setting its $cssclassname property to the name of a style. The properties you define for each style in user.css must be flagged as !important to override the JavaScript Client inline styles.

When you deploy your application on the Omnis App Server, you must put your custom 'user.css' file in the 'html/css' folder on the server.

Customizing the JavaScript Working Message

You can change the appearance and positioning of the working message displayed in the JavaScript Client when some processing is occurring. The working message is an animated GIF which is in a DIV laid over the main JavaScript Client area. You can restyle the working message overlay DIV by adding your own class in the 'user.css'. The default style for the working message is in the omnis.css style sheet:

```
#omnis_overlay  {
    background:url('images/xajax-loader.gif') 20px 20px no-repeat;
}
```

But you can override this in user.css, for example:

```
#omnis_overlay {
    background-position:center;
}
```

Which will center the working image, and/or you could also add your own animated GIF.

Tab Controls

Tab Menu Colors

There are some new properties to allow you to control the colors for the Tab menu in the JavaScript Tab Control.

- **$tabmenubackcolor**
  Used when $tabbackstyle is kJSTabsBackStyleColor. The color of tab menu lines which are not hot. kColorDefault means use $tabbackcolor
- **$tabmenubackiconid**
  Used when $tabbackstyle is kJSTabsBackStyleImage. The icon id of the tab menu line background image for tab menu lines which are not hot. Zero means use $tabbackiconid
- **$tabmenutextcolor**
  The text color used for tab menu lines which are neither hot nor disabled. kColorDefault means use $textcolor
- **$tabmenudisabledtextcolor**
  The text color used for disabled tab menu lines. kColorDefault means use $disabledtabtextcolor
- **$tabmenuhotbackcolor**
  Used when $tabbackstyle is kJSTabsBackStyleColor. The color of the tab menu line background for an enabled line with the mouse over it. kColorDefault means use $hottabbackcolor
- **$tabmenuhotbackiconid**
  Used when $tabbackstyle is kJSTabsBackStyleImage. The icon id of the tab menu line

background image for an enabled line with the mouse over it. Zero means use $hottabbackiconid

- **$tabmenuhottextcolor**
  The text color used for an enabled tab menu line with the mouse over it. kColorDefault means use $hottabtextcolor

Omnis VCS

The following enhancements have been added to the Omnis VCS.

Project Folders

You can create *folder classes* in an Omnis library to allow you to store and organize the classes within the library. When you check a library into the Omnis VCS these folder classes are copied into the VCS project. It is now possible to create a folder class directly within a VCS project via the hyperlink option 'New Folder' available at the VCS project level. This will allow you to organize the classes in your VCS project, but will also allow you to organize non-Omnis Objects (that is, objects that are not Omnis classes) such as external components.

Depending on how a project folder is created and what it contains, it can have one of three possible states:

- a "*normal"* folder is one that is generated within an Omnis library and contains only Omnis classes
- an "*external"* folder is one that is generated from within the VCS
- a "*hybrid"* folder is one that contains both Omnis classes and non-Omnis Objects

Building a Project

When you build a project, *normal* folders will be built in the destination library as at present. *External* folders will be built into the file system using the folder name as a directory appended to the build path with any non-Omnis Objects built there. A *hybrid* folder will build a folder class in the Omnis library along with any Omnis classes inside it as well as building to the file system.

Moving classes & components between folders:

If you move a non-Omnis Object to a folder class that was previously generated from an Omnis library (a *normal* folder), that folder will become a *hybrid* folder. The same thing will happen when you move an Omnis class to an *external* folder.

Note: Once a folder has become a *hybrid*, it will remain one: Therefore empty folder classes may be generated when building a project even if you have deleted all classes from the folder.

Hiding and Showing Project Folders

It is now possible to hide the folders within a VCS project thereby giving you a flat view of all the components in your project. This may be useful when you want to quickly locate a class rather than navigating the entire folder structure in your project.

To hide or show the folders in your project, click on the Hide / Show folders hyperlink option in the VCS Browser.

Showing Checked Out Classes

It is now possible to view only the checked-out classes from a VCS project. To show the Checked Out classes in your project, click on the Show Checked Out hyperlink option in the VCS Browser. To show all classes click on the Show All Classes option.

This option may be useful when you want to quickly identify all the classes that are checked out from a project, and you can combine this option with the ability to hide/show project folders, as above, to view all checked out classes within folders.

When the Checked Out classes are showing you can select one or more classes, right-click on them and select the Check-in option: note you can only check in classes for a single project at a time. You can also run the Class Comparison tool from the same context menu. If there are classes that belong to more than one library, or if a library is not open, these options are disabled.

Version Number Check-in Preference

There is a new VCS preference “Enable version number validation” that allows you to disable version number checking when checking objects into the VCS. The new preference is available in the VCS Options on the Check-in tab.

Pre-Studio 5 VCS Repositories

There is no change in the structure of VCS repositories from Omnis Studio version 5 to version 6, but if you are upgrading to Omnis Studio 6 from a version prior to version 5, you cannot use your old VCS repositories. In Omnis Studio version 5 there were a number of significant changes to the structure of the VCS which means repositories created in versions of Omnis Studio prior to version 5 will not work with Omnis Studio 6.

Existing VCS repositories created in versions of Omnis Studio prior to version 5 must therefore be re-created in Omnis Studio 6 to ensure that they are in the new format. To do this, you must do a build of your existing project (or projects) using your old version of Omnis Studio, create a new VCS repository in Omnis Studio 6, and check in your project(s) and components into the new Studio 6 repository. You will also need to setup all user accounts and preferences in the new repository.

Miscellaneous Enhancements

The following section describes various enhancements or updates in Omnis Studio 6.0: most of these enhancements relate to the JavaScript Client or components.

Adding Objects to JavaScript Forms

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

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

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

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

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

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

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

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

Remote Form Instance Group

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

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

containing a subform object named sfobj, with a classname of RFSUB, then after both forms have constructed:

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

Debugging Methods

Breakpoints

The behavior of Breakpoints in the Omnis debugger has changed in this version. The debugger no longer switches back automatically to the Omnis application window when it encounters a breakpoint in code executing in the JavaScript Client. In this case, when a breakpoint is encountered, the Omnis entry (button) in the Windows Task bar will flash (the default color is orange) and you will have to click on the button to return to the Omnis application window to continue debugging.

Trace Log

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

Alternatively, in JavaScript client-executed methods you can use the *Send to trace log* command which sends the text to the JavaScript console (provided it is available).

Defining a List from a SQL Class

You can use the $definefromsqlclass() method to define a list variable from a schema, query, or table class. The definition for $definefromsqlclass() has been simplified and is as follows:

```
$definefromsqlclass(class[,row,parameters])
```

Where *class* is a Schema, Query, or Table class (name or item reference to it), and the *row* and *parameters* are optional.

The *row* parameter affects the columns used when the SQL class is a schema or table referencing a schema. A row with no columns (or the parameter is omitted) means that the list is defined using all the columns in the schema. Otherwise if the *row* is specified each column in the *row* becomes the name of a column to add to the list definition from the schema. The *parameters* can be a list of parameter values that are passed to $construct() of the table class instance created by the method.

For example:

```
Do list.$definefromsqlclass('schema',row('c1,'c2'))
```

would only include columns c1 and c2 in the list definition.

```
Do list.$definefromsqlclass('schema')
```

Would include all the columns in schema.

To include all columns and call $construct with parameters:

```
Do list.$definefromsqlclass('table',row(),1,2,3)
```

This method passes parameters 1, 2, 3 to $construct and includes all the columns from the schema.

Note the new method parameters are backwards compatible with the old calling conventions in previous versions.

List and Row Columns

The number of possible columns in list and row variables has been increased from 400 to 32,000. You should be aware that the limitations on memory may limit the number of rows in lists with many columns.

Calling Private Methods

There is a new Omnis function called *callprivate()* that allows you to call a private method within the current class or instance and return a value. The syntax is:

```
callprivate(method[,parameters...] ;; calls the private method
```

The function can be called in client methods in the JavaScript Client.

Default Web Browser

The Omnis preference $jswebbrowser (introduced in Studio 5.2) has been renamed $webbrowser and specifies which browser the Test Form option uses for displaying JavaScript remote forms (you can edit the Omnis preferences via Tools>>Options).

Note that if the Web Client plug-in functionality is installed in the tree (by default it is not present in Studio 6.0) the $webbrowser property becomes $pluginwebbrowser and specifies the default web browser for testing Web Client plug-in based remote forms.

Setting the Current Field

You can use the $setcurfield() method in a remote form instance to place the cursor in a specified field. There is a new second parameter for the method that allows you to select all of the contents of the field (if the control supports content selection).

```
Do $cinst.$setcurfield(vNameOrIdentOrItemref[,bSelect=kFalse)
```

sets the current field on the client and if bSelect=kTrue and if supported by the control all of its content will be selected; executing $setcurfield('') will remove the focus from the current field as in previous versions. You can specify the field by name, ident, or item reference also as before.

New Page Browser Prompt

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

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

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

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

Navigation Bars

The JavaScript Navigation Bar control has some new events to allow you to detect when the push or pop animations in the navbar have completed. The events evPushFinished and evPopFinished are triggered when the push or pop animations complete. Both events have one event parameter which is the associated page number that has been pushed or popped.

File Upload Dialog

In the JavaScript File control you can now close the file upload dialog programmatically by assigning kJSFileActionCloseUpload to $action. Closing the dialog using this action does not generate evFileUploadDialogClosed.

Client Object Parameters

The in the HTML that contains the JavaScript Client is called omnisobject1. It contains various parameters that provide details about your application that are sent to the Omnis App Server when the client connects. Existing users should note that the names of these parameters have changed to be more compliant with HTML5. In effect, the old names have been prefixed with 'data-', and are now all lowercase.

The code for omnisobject1 from the 'jsctempl.htm' file is shown below (the template is in the 'html' folder under the main Omnis folder):

```
<div id="omnisobject1" style="position:absolute;top:0px;left:0px"
   data-webserverurl=""                  (was WebServerURL)
   data-omnisserverandport=""            (was OmnisServerAndPort)
   data-omnislibrary=""                  (was OmnisLibrary)
   data-omnisclass=""                    (was OmnisClass)
   data-param1="" data-param2=""         (was param1, param2,..)
   data-commstimeout="0">                (new parameter)
```

Existing users should note that any existing HTML pages containing the omnisobject1 that uses the old parameter names will continue to work against a version 6.0 Omnis App Server, so it's not essential that you update your HTML pages.

Server Timeout

The JavaScript Client now gives the end user the option to timeout a request or carry on waiting. There is a new parameter 'data-commstimeout' in the omnisobject1 within the JavaScript Client HTML template to control the server timeout. The default value of zero means that no timeout is applied, and the client will continue to wait for a response. To apply a timeout, you need to enter an integer representing the timeout in seconds. When the client sends a message to the server it must respond within this timeout, otherwise the user will be prompted to either continue waiting for a response, or abort the request.

Managing Wifi Connections

Managing Wifi connections in the JavaScript Client has been improved. The client should now handle wifi connection going away while processing a request at the server, so when the connection comes back, the client should respond properly. Otherwise, you have the option to provide a timeout, as above.

Combo Boxes

The JavaScript Combo Box now handles a numeric dataname properly, and also has the $negallowed property which means it can display negative numbers.

Tab Count

The JavaScript Tab control now allows you to set $tabcount to zero, and to set $tabcount at runtime using the notation.

Time Zone Functions and Settings

The datetime and timezone functions loctoutc(), utctoloc(), tzcurrent(), tzstandard(), and tzdaylight() are now built into Omnis Studio rather than being external components, but their functionality is the same: they are listed in the 'Date and Time' group in the Omnis Catalog (F9/Cmnd-9). They were previously Java based and dependent on the Omnis Web Services component, but this dependency is no longer the case. This change will allow you to use the functions in the JavaScript Client to convert local times to UTC since the client and server need to both use UTC time.

You should note that on 32-bit Windows the TZ codes returned by the timezone functions are the long time zone names and not the short abbreviated time zone names. If you want to use the short time zone names, you can add a mapping to studio.stb, from the full name to the abbreviation you wish to use.

Time Settings

The Omnis App Server and JavaScript Client exchange dates and times in UTC time, regardless of where your server is located. There is a new remote task property $jslocaltime to allow you to exchange times in Local time rather than UTC. If true, the JavaScript Client and the Omnis App Server exchange date-time values in local time rather than UTC time. $jslocaltime must be set in design mode: it cannot be assigned at runtime.

Justified Report Text

You can now justify report text which means that the text will be aligned to both the right and left edges of the text object or field. You set $align for the report field to kJustifiedJst.

Custom Variable Types

You can define your own custom variable types. To do this you have to create a custom method called $<customatttribute name>.$assign, and then the $type, $subtype and $sublen properties of the custom variable return their value according to the type of parameter 1 of $<customatttribute name>.$assign.

Changing List Values

There has been a change in the way the $assigncols() method is used to update the values in a list. When $assigncols() is assigned to a list row, Omnis now looks up each column name and associates it with the variable of the same name (typically an instance variable) if one exists with the same data type as the column.

In addition, this change also corrects an issue with $cols.$add(). In previous versions, passing a single parameter (which is meant to add the column with the passed variable as its definition) was actually using the content of that variable as the column name. This resulted in an inconsistency between client and server methods, and has now been corrected to use just the name and not the content of the variable. For example, if iVar is an instance variable, then $cols.$add(iVar) will always add column iVar in both server and client methods.

Error Text in JavaScript Client

#ERRTEXT is now available in JavaScript Client methods.

Drag and Drop for JavaScript Components

In design mode, you cannot drag and drop a component from one remote form to another if the forms have different screen sizes (set in $screensize). Dragging and dropping components only works for remote forms with the same screen size setting.

Client Methods

Object Properties

The ability to run methods on the client was introduced in Studio 5.2.1. You should note that some properties can only be assigned in server methods, but you can read the current value of many more remote form properties and component properties in client methods. So, you cannot change the size and position of an object in a client method (by setting $top, $left, $width, and $height), but you can return the current values for an object in a client method, e.g. $cobj.$width returns the width of the current object.

Custom Methods

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

Appendix

Web Client Plug-in

If you wish to use the Omnis Web Client plug-in functionality in your applications, or use the Web Client migration tool to migrate your applications to the JavaScript Client, you need to download and install the Web Client Development kit from the Omnis website (www.tigerlogic.com/omnis). This installer contains all the necessary files to reinstate the Web Client functionality and components, and enable the migration tool to work.

Migration Tool

Omnis Studio 6.x includes a tool (also present in Studio 5.2.x) to allow you to migrate remote forms that use the Web Client plug-in to the new JavaScript Client based remote forms. Due to the many differences between these form types, the migration process is not complete, that is, not all the existing Web Client controls exist in the JavaScript Client, so you will need to update some of the remote form objects and methods yourself. The form migration tool creates a copy of your old remote form class and places the new JavaScript form inside a folder within your library.

The **Migrate Forms** option is available in the Studio Browser when a library is selected. When you click on the option, the migration window will open showing all your open libraries and all the Remote forms in each library. The **Options** menu bar option allows you to setup the migration parameters, including the location of the new remote form classes (a folder in your library called JSFormsFromMigration), the location of the migration log files (default is JSMigrationLogs folder in the main Omnis folder), and the mapping of Web Client controls to the new JavaScript controls.

Control Migration Mapping

The migration process generates a new remote form with the equivalent JavaScript controls, if they exist. If there is no corresponding JavaScript control, the migration tool creates a place holder control in order that any methods associated with the original Web Client control are not lost. For example, the Button Area control does not exist for JavaScript remote forms, therefore any Web Client Button Areas are migrated to standard JavaScript buttons and the original method from the button area is placed behind the new button. Heading lists are migrated to Data Grids, while Icon Arrays and Sidebars are migrated to standard List fields.

The **Objects** tab on the Options window allows you to change the control migration mapping, but for most cases you should use the suggested migration mapping and then modify the place holder controls manually. In some cases, you may need to significantly change the controls and methods in your new Remote forms in order for them to function correctly in the JavaScript Client.

$enablesenddata Property

In older versions of Omnis Studio, the $senddata() method could be used to control when form data was sent to remote forms displayed in the Omnis Web Client; the $enablesenddata remote task property was introduced to allow you to enable or disable the $senddata() method. However, the new JavaScript Client handles when the form data is sent to the client automatically, so the $senddata method is not required. If the $enablesenddata property is enabled in any of the remote tasks in your old Web Client based application, the migration tool will disable it and add a note to the log. If you try to open a new JavaScript form controlled by a remote task with $enablesenddata turned on there will be an error on the client; the property must be disabled in the remote task for JavaScript Client based remote forms to work.