

What's New in Omnis Studio 5.1.1

Extending Omnis to New Platforms and New
Markets

TigerLogic Corporation

April 2011

14-042011-01

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

© TigerLogic Corporation, and its licensors 2011. All rights reserved.

Portions © Copyright Microsoft Corporation.

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

© 1999-2011 The Apache Software Foundation. All rights reserved.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

Omnis® and Omnis Studio® are registered trademarks of TigerLogic Corporation.

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

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

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

ICU is Copyright © 1995-2003 International Business Machines Corporation and others.

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

J2SE is Copyright (c) 2003 Sun Microsystems Inc under a license agreement to be found at:

<http://java.sun.com/j2se/1.4.2/docs/relnotes/license.html>

Portions Copyright (c) 1996-2008, The PostgreSQL Global Development Group

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

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

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

Acrobat is a registered trademark of Adobe Systems, Inc.

CodeWarrior is a trademark of Metrowerks, Inc.

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

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

This software is based in part of the work of the FreeType Team.

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

Table of Contents

ABOUT THIS MANUAL	5
WHAT'S NEW IN OMNIS STUDIO 5.1L.....	6
IOS CLIENT	7
SQL QUERY BUILDER	8
TOOLBARS.....	18
ULTRA-THIN CLIENT.....	18
WHAT'S NEW IN OMNIS STUDIO 5.1.....	19
CREATING OMNIS APPLICATIONS FOR IOS	20
AMAZON DAM	74
MISCELLANEOUS ENHANCEMENTS.....	87
WHAT'S NEW IN OMNIS STUDIO 5.0L.....	88
WINDOWS 7	89
MAC OS X	89
CLIENT COMMANDS	90
ENTRY FIELDS	91
REMOTE FORMS	91
WINDOW FIELDS	91
LOCALIZATION	93
STUDIO MESSAGES.....	93
TRACE LOG	93
WEB COMMANDS	93
SERVER LOAD SHARING.....	94
USING THE ODB	95
WHAT'S NEW IN OMNIS STUDIO 5.0	97
NEW FEATURES	97
SERIAL NUMBERS.....	99
LIBRARY AND DATA FILE CONVERSION	99
SYSTEM REQUIREMENTS	99
WINDOWS MOBILE CLIENT	100
REMOTE MENUS.....	118
WEB SERVER PLUG-IN	120
UNICODE	124
LOCALIZATION	136
OMNIS VCS.....	148
METHODS.....	152
FORM COMPONENTS.....	154
ACCESSING THE WINDOWS REGISTRY.....	158

Table of Contents

DAMS	161
MISCELLANEOUS ENHANCEMENTS.....	167
SECURE WEB COMMUNICATIONS	175
FUNCTIONS.....	179

About This Manual

This document describes the new features and enhancements in Omnis Studio 5.1.1. It also contains information about Omnis Studio 5.1, 5.0.1 and 5.0 which was published in each corresponding ‘What’s New’ manual, but is reproduced here for your convenience.

Please see the file `Readme.txt` for this release for details of bug fixes and any last minute notes for Omnis Studio 5.1.1.

If you are new to Omnis Studio

If you are new to Omnis Studio you should start by reading the *Introducing Omnis Studio* manual and then the *Omnis Programming* and *Extending Omnis* manuals. All the Omnis Studio manuals are available on the product DVD and to download from the Omnis website (www.tigerlogic.com/omnis).

What's New in Omnis Studio 5.1.1

Omnis Studio 5.1.1 provides support for Client Scripting in the previously released iOS client, which allows you to create more powerful and interactive apps for iOS 4 based devices from Apple, including iPad™, iPhone® and iPod touch®. In addition, there is a new SQL Query Builder and several other smaller enhancements, and a raft of bug fixes.

❑ **Client-side scripting on the iOS client**

Client-side scripting has been enabled for remote forms running on iOS devices. This means that you can execute calculations and other code within the iOS client itself, allowing you to create more powerful and interactive apps for iOS devices

❑ **SQL Query Builder**

The SQL Query Builder lets you quickly build, run and store SQL Queries using a graphical interface. The SQB supports the creation of both simple queries requiring little SQL knowledge and more advanced queries using different joins and clauses

❑ **New parameter for \$add in Toolbar groups**

There is a new parameter on the \$add method toolbar method to control precisely when the toolbar is added: this is to address an issue with toolbar buttons flickering when added to toolbars on Mac OS X using the notation

❑ **New control header in Ultra-thin client**

A new control header "x-omnis-ctrl:" has been added to Omnis Ultra-thin client that can be returned at the start of the content to allow you to turn off the pragma:nocache header

iOS Client

Client-Side Scripting

Client-side scripting has been enabled for Remote forms running on iOS based devices such as iPhone and iPad. This means that you can run certain methods in an iOS based remote form containing calculations, switch statements, event handling, and other programming constructs. This capability was disabled in Omnis Studio 5.1, when iOS app development was added to Omnis, but due to restrictions in Apple licensing being lifted, remote forms in Studio 5.1.1 can now take advantage of client-side scripting.

To enable a method to run on the client, you can Right-click/Cmnd-click on the method and select the 'Execute on Web Client' option. If a method contains code that cannot be executed on the client, Omnis will not allow you to set the 'Execute on Web Client' option. When you have enabled a method to run on the client, it will be shown in pink in the method editor.

See the 'Client Method Execution' section in Chapter 1 of the 'Extending Omnis' manual for further details about client-side scripting.

Disabled text color property for iOS buttons

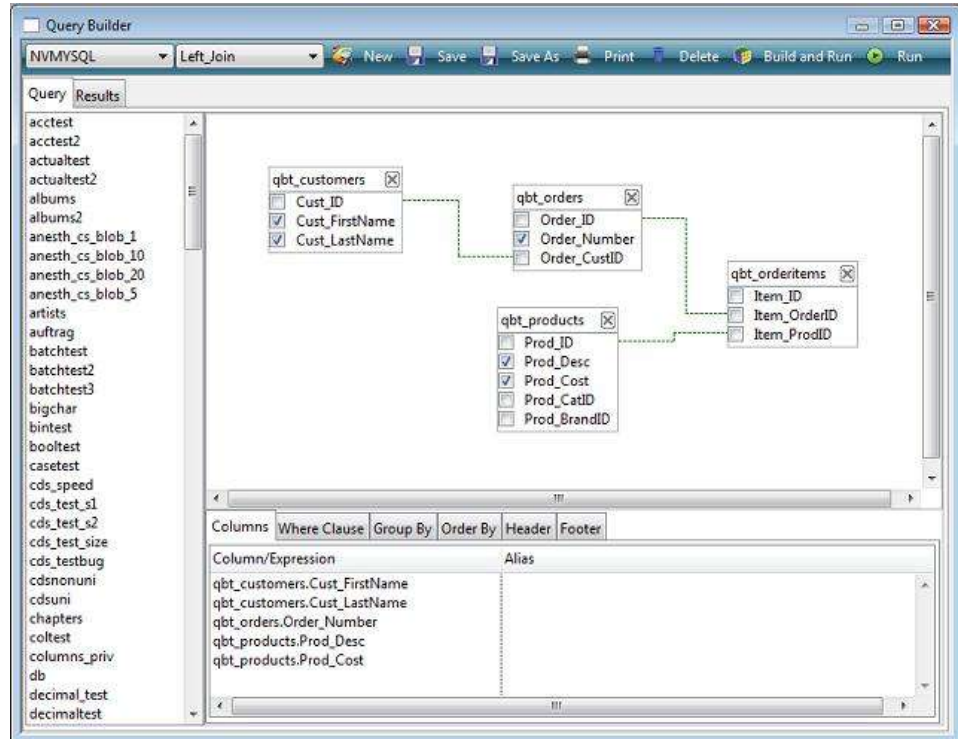
There is a new property for buttons, called `$disabledtextcolor` which is the color used for the button text when the button is disabled. Setting it to `kColorDefault` means `$textcolor` is used. (ST/WT/1702)

SQL Query Builder

The SQL Query Builder (SQB) lets you build, run and store SQL Queries quickly and easily using a graphical interface. The SQB is built into the Omnis Studio IDE and is easily accessed via the SQL Browser. The SQB supports the generation of both simple queries requiring little SQL knowledge and more advanced queries using different join types and clauses. You can save queries for later use and you can create Omnis query classes based on your queries to allow you to take advantage of the queries you build in the SQB in your applications.

SQL Query Builder window

You can open the SQL Query Builder by clicking on the 'Query Builder' option when the SQL Browser option is selected in the main Studio Browser window. The SQB window has three main panes: the table pane on the left showing all available tables, the main design area at the top-right showing the query in graphical format, and the lower tab pane for defining aliases, SQL clauses and expressions.



All panes in the Omnis SQB are resizable and can be saved via 'Save Window Setup' of the main context menu. The list of available tables may also be refreshed at any time via the table list context menu.

Creating a Query

The SQL Query Builder allows you to construct most types of query simply using drag and drop and the context menus available as appropriate in each pane of the main SQB window. In order to construct a query, you must be logged on to a SQL session via the SQL Browser.

The first time you open the SQB the query list will be empty allowing you to create a new query. If an existing query is currently displayed pressing either the 'New' toolbar button or selecting '<new query>' from the query dropdown list will clear the query from the screen.

Selecting a SQL Session

All open sessions are shown in the droplist at the top-left of the SQB window. You can select a session to build your query by dropping down the session list and selecting a session.

Adding a Table

Once a SQL session has been selected a list of tables for the current session is shown in the left hand pane. To include a table in a SQL Query simply drag the table into the main design area on the right.

Removing a Table

You can remove a table from a query either by clicking on the 'X' icon for the table or by opening the context menu for the table and selecting 'Remove Table'. You have to confirm if you want to remove a table from the current query.

Refreshing a Table

If a table has been modified outside the SQB, you can refresh the table in the SQB by right-clicking on the table and selecting 'Refresh Table' from the context menu. This option will add any new columns or remove any deleted columns to reflect the current state of the server table.

Selecting Columns

Once a table has been added, you can select columns by selecting their names individually or by right-clicking on the table you can check or uncheck all the columns in the table. Alternatively, you can specify that the SQB checks all columns automatically when a table is added using the 'Select all Columns on drop' option within the 'Options' dialog (see the Options section).

Adding Column Aliases

You can add an alias for each selected column in the 'Columns' tab of the lower tab pane by clicking in the 'Alias' column of the current line.

Columns	Where Clause	Group By	Order By	Header	Footer
Column/Expression	Alias				
qbt_customers.Cust_FirstName	Firstname				
qbt_customers.Cust_LastName	Lastname				
qbt_orders.Order_Number	Order Number				
qbt_products.Prod_Desc	Product				
qbt_products.Prod_Cost	Cost				

You can reorder columns in this pane by dragging and dropping column names in the list.

Creating Joins

To construct your queries, you can create joins between tables either using drag and drop in the main SQB design area or using the Table Joins dialog. To create a join, drag a column name from one table and drop it onto the column name within the table you wish to join with. Alternatively, you can right-click on a table to open the 'Table Joins' dialog from the context menu.

Table A	Column A	Operator	Table B	Column B	Join Type
qbt_orders	Order_CustID	=	qbt_customers	Cust_ID	Left
qbt_orderitems	Item_OrderID	=	qbt_orders	Order_ID	Left
qbt_products	Prod_ID	=	qbt_orderitems	Item_ProdID	Left

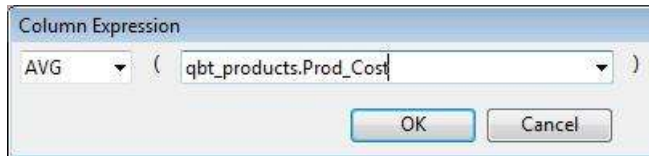
Join		Join Type	
Table.Column A	qbt_orders.Order_CustID	<input type="radio"/> Where Clause	<input type="radio"/> Right
Table.Column B	qbt_customers.Cust_ID	<input type="radio"/> Inner	<input type="radio"/> Full
Operator	=	<input checked="" type="radio"/> Left	
		<input type="button" value="Add"/> <input type="button" value="Update"/> <input type="button" value="Delete"/>	
<input checked="" type="checkbox"/> Show all joins		<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

The 'Table Joins' dialog lets you modify joins, set the operator and type, re-order using drag and drop and switch columns using the context menu.

Joins may also be deleted via the context menu of the joined column in the main design area and Line and style preferences can be set via the 'Options' dialog.

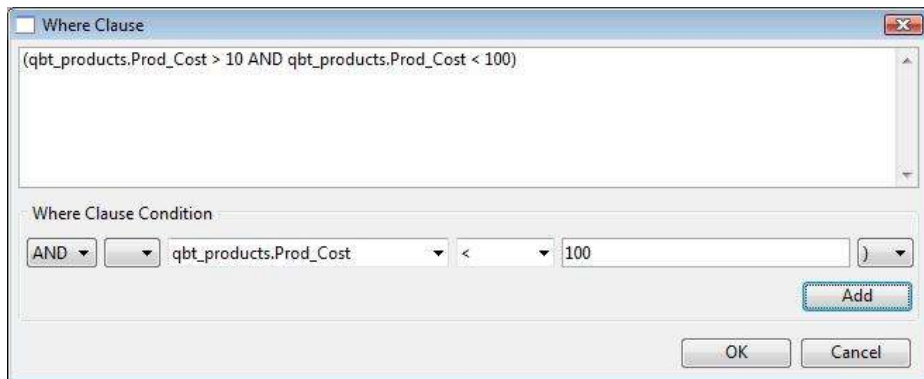
Adding Column Expressions

In addition to specifying aliases, the 'Columns' tab of the lower tab pane lets you add expressions to a query via the context menu. Selecting 'Add Expression' from the context menu opens a dialog where you can add common SQL expressions such as AVG, COUNT, MAX, MIN and SUM.



Adding a Where Clause

You can add a Where clause condition by dropping a column from a table onto the 'Where Clause' pane of the lower tab pane. When you drop a column the 'Where Clause' dialog is opened automatically and the column is pre-selected. You can also right-click on the Where Clause pane to Edit the column conditions for the current query.



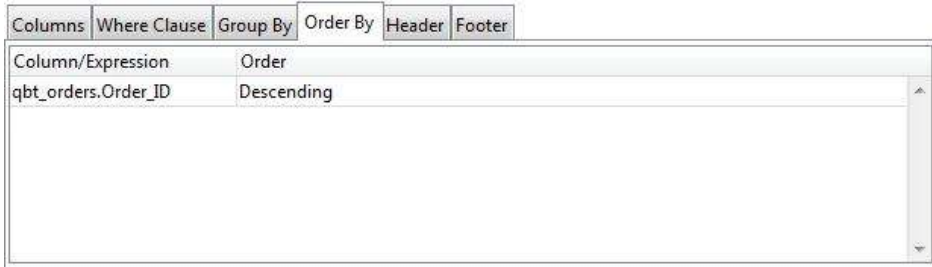
Adding a Group by Clause

You can add a Group By clause, to group selected rows together to return a summary of information, by dropping a column name onto the 'Group By' pane of the lower tab pane. You can also add a Having Clause to restrict the rows used by the Group By clause either by dropping a column from a table or via the context menu.



Adding an Order by Clause

You can add an Order By clause by dropping columns onto the 'Order By' tab of the lower tab pane; when you drop a column Descending order is selected by default but you can change it to Ascending. You can add Expressions to the Order By clause using the context menu by right-clicking on the Order By pane.



Modifying the SELECT Construct

The 'Header Tab' in the lower tab pane lets you enter an alternative construct, such as 'SELECT DISTINCT' and where supported 'SELECT TOP 100'. This tab also lets you add comments to precede the generated SQL Statement.



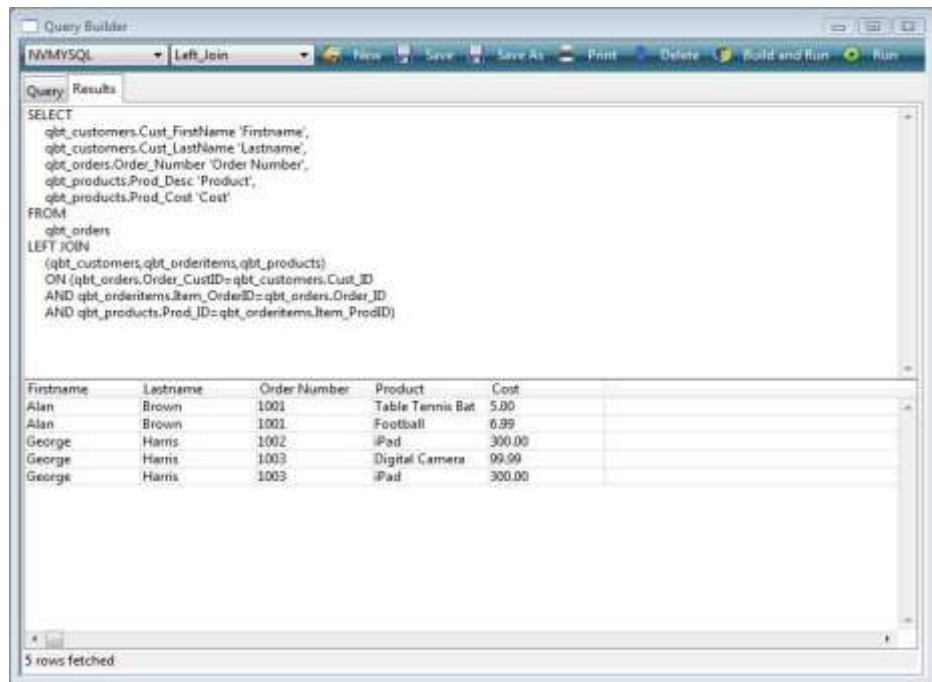
Adding Extra Query Text

The 'Footer Tab' in the lower tab pane lets you add any additional query text to be appended to the generated SQL Statement.



Running a Query

You can run the current query from the main toolbar in the SQB window using the 'Build and Run' or 'Run' option. Both buttons switch the main pane to the 'Results' pane showing the SQL script and results generated by the Query. You can make changes to the SQL script generated if required and the query can be executed again using the 'Run' button. Note that using the 'Build and Run' option will rebuild the statement from the saved query text and therefore overwrite any changes you may have made.



Any errors which occur are reported in the status bar. If the full error text is not displayed, you can click on the status bar to open a dialog showing the full error text.

Saving a Query

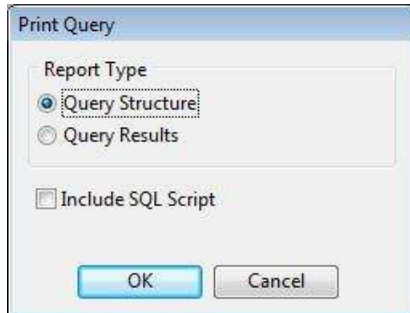
You can save a query using the 'Save' or 'Save As' toolbar option. When using 'Save' option for the first time, or the 'Save As' option, you can add a description for the query. The description is also available to view/edit via the 'Query Info' option of the main context menu. Once saved, a Query is added to the list of Queries available in the dropdown list in the main SQB toolbar.

Deleting a Query

The Delete button in the SQB toolbar button lets you delete the currently selected query (you must confirm the deletion).

Query Reports

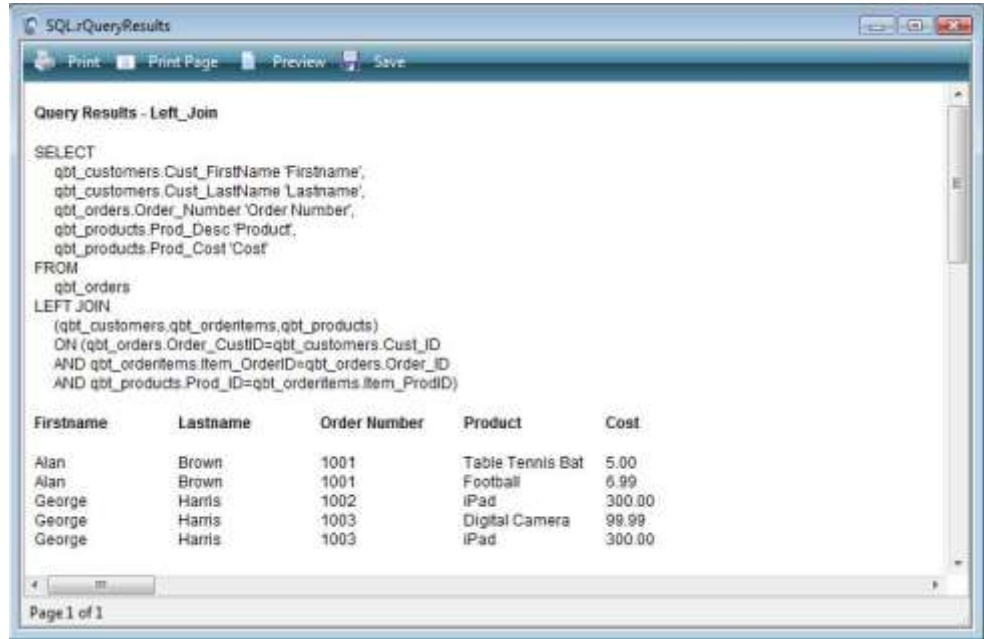
There are two types of report available via the Print button on the main toolbar or the context menu opened by right-clicking on the Query design pane: both these options open the Print Query dialog.



The Print Query dialog allows you to print either the Structure or the Results of the current query. In addition, you can include the generated SQL Script from the last executed SQL query in both reports.

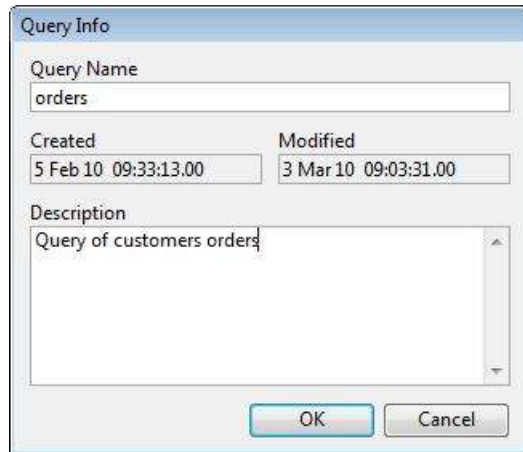
Query Results Report

The 'Query Results' report shows the results of the last executed SQL query. Column widths reflect those of the results pane and may therefore be adjusted prior to printing.



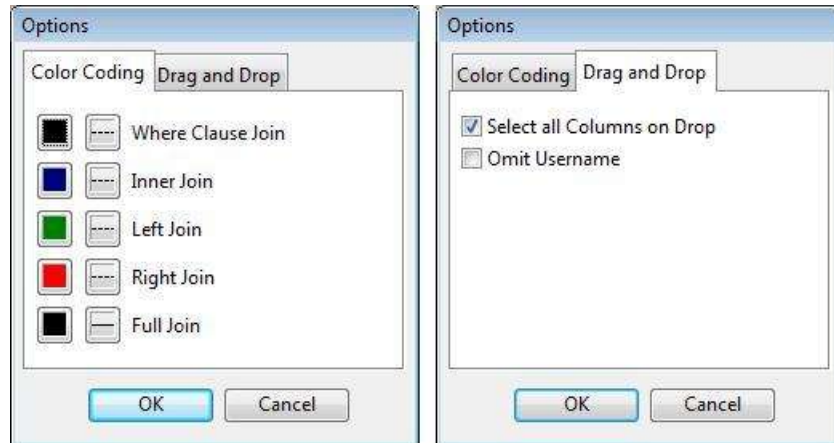
Query Info

The 'Query Info' dialog available from the main query context menu displays information about the current query. You can change the name and description of query.



Options

You can set various options for the SQL Query Builder in the 'Options' dialog which is available from the context menu on the main query window. You can specify the line and color styles for joins, and you can set preferences for dragging and dropping during table creation.



For databases where table names are prefixed by a username, the Omit Username option is particularly useful when running the same query against different databases where the username is different, but the table and column names are the same.

Creating a Query Class

You can create an Omnis Query class based on the current quer. To do this, you can drag the current query from the main query design window on the right and drop it on to an open library in the main Studio Browser. When you create a query class in this way, all the additional query text is added to the class, together with any associated Omnis Schema classes in order to map to the server tables in the query. Note that this feature is restricted by the same limitations as a query class and therefore only supports 'Where Clause' joins.

Creating a Statement Block

You can copy the generated SQL Script to the clipboard in a 'Begin Statement', 'Sta:', 'End Statement' block by right-clicking on the 'Results' pane and selecting 'Create Statement' from the context menu. You can paste the statement into an Omnis method, providing an alternative method of executing your query in an Omnis Studio library where a query class is not appropriate.

Toolbars

Adding buttons to Toolbar groups in OS X

There was an issue with toolbar buttons flickering when added to toolbars on Mac OS X using the notation (reported in ST/WC/461).

To address this issue, the `$add` method (and `$addafter` and `$addbefore`) has a new optional second argument: `$add(cClassname/rItem[,iOSXmode])` inserts the toolbar into the window docking area.

The `iOSXmode` parameter allows you to control precisely when the toolbar is added: for Mac OS X style toolbars, omit `iOSXmode` to change the docking area now; if `iOSXmode=1` defer changes until a call with `iOSXmode=2`.

Ultra-Thin Client

New control header in Ultra-thin Client

A new control header "x-omnis-ctrl:" has been added to Omnis Ultra-thin client that can be returned at the start of the content to allow you to turn off the `pragma:nocache` header. (ST/WT/1696)

When returning content, it can now either start with "content-type:" (as in earlier versions of Omnis) or "x-omnis-ctrl:". If `x-omnis-ctrl:` is present, it must be at the start of the content, to have any effect. To prevent the Omnis web server plug-in from generating the `pragma:nocache` header, start the content with

```
x-omnis-ctrl:nopragma<cr><lf>
```

The letters can be in any case, but there must be no white space.

What's New in Omnis Studio 5.1

Omnis Studio 5.1 provides support for mobile devices running Apple's iOS (formerly known as iPhone OS). This mobile platform originated on the iPod and iPhone, but is now on the iPad from Apple.

❑ **Apple iOS support**

Studio 5.1 allows you to create remote forms that will run on iOS based mobile devices which means you can deploy your Omnis applications to the latest devices from Apple including iPad™, iPhone® and iPod touch®.

❑ **Amazon DAM**

A new Omnis DAM, available for Windows, Mac OS X, and Linux, that allows you to access the SimpleDB from Amazon, a non-relational data store widely used in e-commerce applications and large information websites

❑ **HTTP commands**

The Map+ parameter has been added to the HTTPGet() and HTTPPost() commands to specify that plus (+) characters in CGI parameter names and values in the CGI List are URL encoded using hex.

❑ **Apple Menu Hide Key String**

The AppleMenuHideKey command (Apple + H key combination) has been added to the built-in strings in Omnis, so you can edit this string to change the letter to activate the command.

❑ **Remote Menu Lines**

You can now set the text for a remote menu line to \$st.id. The string table lookup occurs when the menu is built on the client, before any event processing for the menu.

Library Conversion

Omnis Studio 5.1 will try to convert libraries created in a previous version of Omnis Studio when you open them in the new version. Please ensure you have a secure backup of all libraries before you open them in Omnis Studio 5.1.

Serial Numbers

You will require a new serial number to run Omnis Studio 5.1. Contact your local sales office for further details.

Creating Omnis Applications for iOS

Introduction

To create an iOS app using Omnis Studio, you have to create a *Remote Form* specifically designed for iOS. Once you have switched an Omnis remote form to an iOS form and placed iOS components on the form, it cannot be converted back to a non-iOS remote form for the web, or used on other mobile platforms such as Windows Mobile. However you can add iOS remote forms to the same Omnis application (library) containing Web and Mobile remote forms and re-use the underlying methods and schema classes, for example, in your iOS remote forms.

Many of the techniques required for building iOS apps, and in particular for creating Omnis Remote Forms, are the same as those for creating Omnis Web and Mobile Client applications available in previous versions. These techniques are discussed in detail in the *Extending Omnis* manual for Omnis Studio 5, available to download from the Omnis website. However there are several features that are unique to the iOS support which are covered in this chapter.

iOS naming

General references to “iOS” or an “iOS device” usually refer to any device running the iOS from Apple, including the iPad, iPhone and iPod touch. However, you should check the capabilities of each device and test your iOS app on all devices you wish to support in your app.

Requirements

Development Software

To create Omnis applications for iOS you must use Omnis Studio 5.1 or above. You can design remote forms for iOS in the Mac OS X, Windows, or Linux version of Omnis Studio. That is, the iOS remote form components are available in all platforms, but you will need a Mac OS X computer to build your app.

To build device and simulator apps, and to run the latest version of Xcode and the SDK from Apple, you will need Mac OS X 10.6 (Snow Leopard) or above.

You will also need a copy of iTunes to sync your computer with your iOS device.

iOS and supported devices

To use the iOS functionality in Omnis Studio, the client will need an iOS based device. The iOS runs on various mobile or handheld devices from Apple including:

- iPhone®
- iPod touch®
- iPad™

The majority of iOS features available in Omnis Studio 5.1 will work on all iOS enabled devices, but you will need to check the specification of individual devices and take into

account the different screen sizes for different devices when designing your app. We strongly urge you to test your Omnis iOS apps on all the devices you wish to support before deploying your application.

iOS Developer Program and Distribution

You need to sign up to the Apple iOS Developer Program to obtain the necessary files from Apple and to build your Omnis iOS app. To distribute your iOS app, within your organization or at a client site, you need to use one of the approved Distribution methods which usually will allow you to deploy your iOS app to a specified number of devices. Further details about distribution are available from Apple Inc, and are outlined later in this chapter.

Creating Remote forms for iOS

Designing remote forms for iOS is more-or-less the same as designing standard remote forms for the web or any other platform in Omnis Studio. However, remote forms for iOS use the native iOS controls so you need to “switch on” the iOS functionality and components in the remote form, using the \$ios remote form property.

If you open an existing remote form, containing standard non-iOS specific components, the \$ios property will be grayed out and you will not be able to switch the remote form to an iOS enabled form. Similarly, an iOS enabled remote form that contains iOS specific components cannot be switched back to a standard web remote form. For this reason, you will have to create your iOS remote forms from scratch, since the iOS components and standard web components are not interchangeable.

To create a remote form for iOS

- Click on the **New Class** option in the Studio Browser, scroll down and click on the **iOS Form** option

This option will create a new remote form with the \$ios property already set to kTrue, plus the default screen sizes for the form are assigned for each orientation.

Or you can

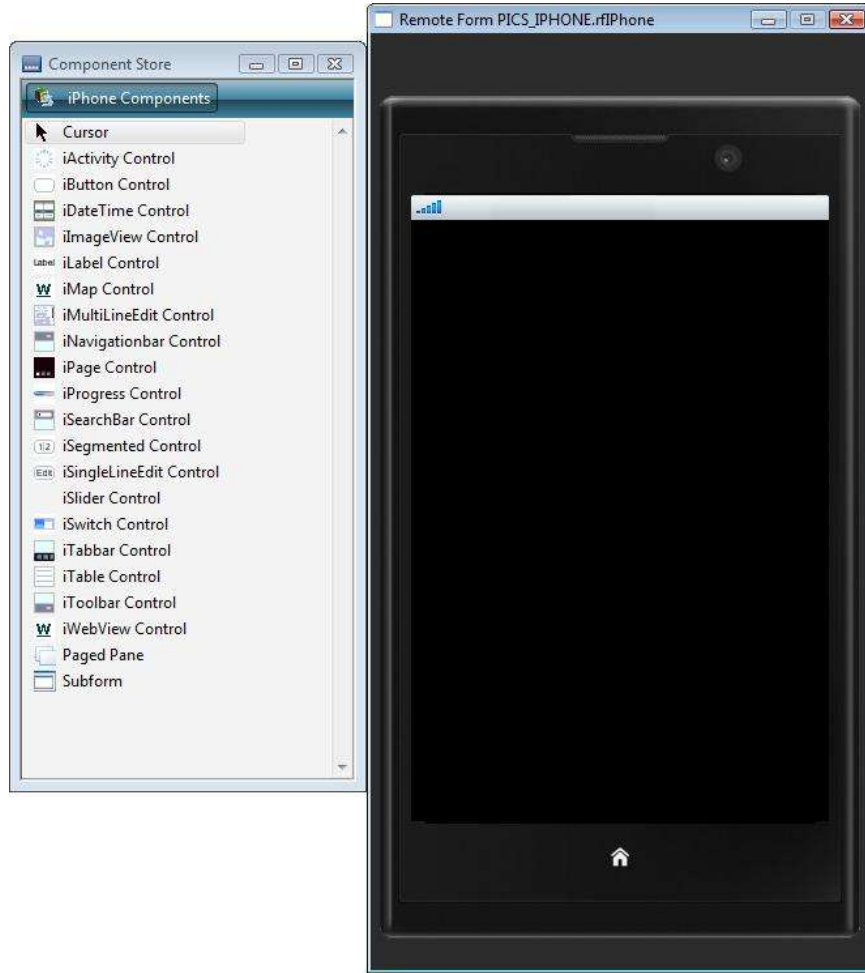
- Create a new remote form using the **New Class>>RemoteForm** option in the Studio Browser
- Click on the background of the remote form and in the Property Manager set its \$ios property to kTrue

Plus, for either option, you’ll need to:

- Create a remote task and assign its name to the iOS remote form’s \$designtaskname property

When you create or switch a remote form to iOS, the Component Store will display iOS Components only, under the ‘iOS Components’ group, replacing the standard Web Components, while the remote form design window will display a mobile device image in

the border and the title bar. The iOS components are located in the ioscomp folder in your Omnis development tree.



The iOS remote form (right) and the Omnis Component Store (left) showing the twenty or so iOS components

From here on, existing Omnis developers will find the layout and visual design of forms for iOS very familiar, although you set the behavior of components in a different way. You can design your remote form for iOS in the same way as you would for a standard web-based remote form.

When you open a remote form from the Studio Browser, the Component Store will display the appropriate components depending on the type of remote form you open, either a remote form for iOS, or for web browsers, or some other client device.

There are several properties you need to set to specify the behavior, event handling and other functionality in your iOS remote form; these are listed in the *iOS Form Properties and Methods* section below.

Screen size and form layout

By default, the `$screensize` property is set to `kSSZiOS320x480Portrait`, but you can switch the form to `kSSZiOS320x480Landscape` to design an alternative layout. This screen size is supported on the iPhone and iPod touch, while the `kSSZiOS768x1024...` screen sizes are for iPad.

You can design different layouts for the different sizes and orientations and *store them in the same remote form*. In other words, each layout uses the same set of fields (and methods) and the remote form class stores the position of the fields for each screen size/orientation setting. If you have created a form layout for portrait and landscape and the end user tilts the device, the layout of the Omnis remote form will change automatically (unless you have set the portrait only property in the client configuration file).

Window title bar

When designing for different screen sizes you must take account of the iOS title bar. For example, the screen size of an iPhone is 320 x 480 pixels, but by default the title bar at the top of the remote form is visible, giving you a possible screen area of 320 x 460 pixels in portrait mode (the title bar is 20 pixels high). The `$designshowmobiletitle` remote form property (under the Appearance tab in the Property Manager) lets you hide or show the window title on the iOS device.

Form width and height

The `$width` and `$height` properties of the form are independent of the current screen size and orientation, so you can change their values if required, taking into account whether or not the title bar is visible. The settings of `$width` and `$height` for each size/orientation you set up are stored in the remote form class, along with the position of all fields and other components for each layout.

You will notice that the area defined by `$width` and `$height` is shown as a black area and indicates the window area visible on the client; the area beyond the visible area is dotted. You cannot place fields beyond the area defined by the `$width` and `$height` properties using your mouse, although you can set the value of `$left` and `$top` for an object (either in the Property Manager or using the notation) to place any object outside the visible form area.

The default values for the iPhone `$screensize` setting are `$width = 320` and `$height = 460` for Portrait mode, and `$width = 480` and `$height = 300` for Landscape mode. Note this takes account of the window title bar (which is visible by default) so if you hide the design window title bar, you'll need to add 20 pixels to the `$height` setting.

Component Methods and Client Execution

The properties and behavior of the iOS specific components are a little different from the standard remote form components (listed under Web Components in the Component Store), and using the iOS components requires a slightly different approach. The biggest difference between remote forms for the web and iOS is that client method execution is *not allowed* in remote forms in iOS apps. As a consequence, iOS components do not have specific methods (except for \$redraw), rather you control the behavior of objects by assigning values to their properties using the Do <property>.\$assign() method or the *Calculate* <property> as *value* command. The properties for each component, together with code examples, are listed in the *iOS Components* section below.

Debugging Methods

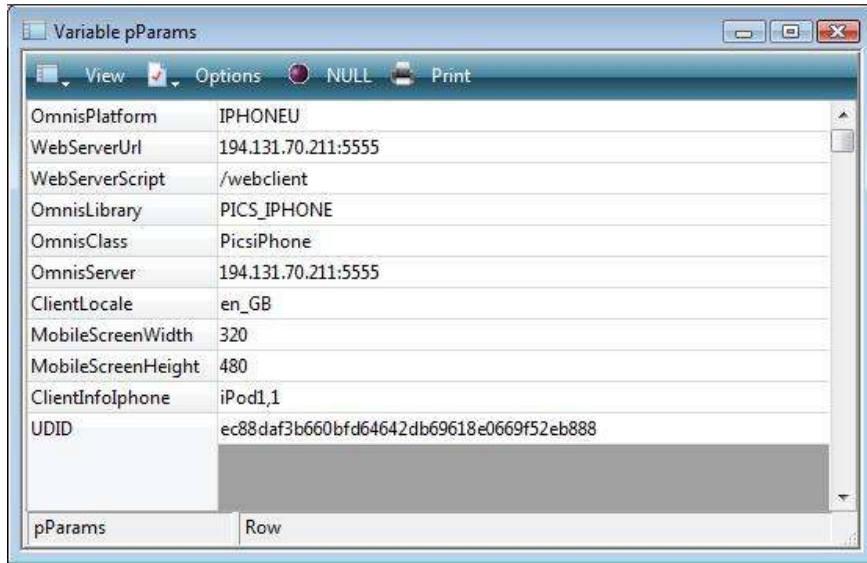
You can debug your methods as you are testing your application on the client device by setting breakpoints in your code. You can right click on a method line and select the Breakpoint option to set a breakpoint. When you test the application on your client device and the breakpoint is encountered in your code, method execution is halted at the breakpoint and the application is temporarily suspended on the device. At this point, you can switch back to Omnis on your development computer and step through the live code, inspecting variable values (Right-click on the variable name anywhere in your code and select the Variable <varname> option), commenting and uncommenting code (using Ctrl-; and Ctrl-'), and so on. When execution has completed in the Omnis method editor, the app will resume operation on the client device.

Getting App and Device Information

`$construct()` for remote forms and remote tasks is passed a row variable parameter, typically called `pParams`. To use `pParams`, you must declare it in the parameters tab of the variables pane for `$construct()`. The row variable contains the connection parameters for the remote form and/or remote task and may contain useful information for your application. The `pParams` variable has the following columns:

pParams column	Description
OmnisPlatform	The client platform, e.g. IPHONEU
WebServerUrl	The URL of the Web Server through which your iOS client application connects to the Omnis Server; during development this will be the IP address of your computer and the port number of your development copy of Omnis
WebServerScript	The path or location of the Omnis Web Server plug-in, usually located in your cgi-bin or scripts folder, e.g. /cgi-bin/omnisnph-cgi.exe; during development, this will be /webclient which means Omnis will use its own built-in web server
OmnisLibrary	The name of the Omnis library containing your iOS app
OmnisClass	The name of the remote form in your Omnis library
OmnisServer	the IP address and port number of the Omnis Server; during development this will refer to your local computer IP address and local copy of Omnis Studio
ClientLocale	The locale of the client device, e.g. en_GB
MobileScreenWidth	The screen width of the current client device
MobileScreenHeight	The screen height of the current client device
ClientInfoIphone	(iOS forms/tasks only) the iOS device type, e.g. 'iPod1,1' for a first generation iPod
UDID	(iOS forms/tasks only) The unique ID of the current iOS device

To view the `pParams` row variable in design mode, you can set a breakpoint in the `$construct()` method and open/test your remote form. Switch to the Omnis debugger, click on the Parameters pane in the method editor, right-click the `pParams` variable and select the 'Variable `pParams...`' option. A small popup window will show the columns in the `pParams` row variable.



Device type

For iOS remote forms or associated remote tasks, the ClientInfoIphone parameter contains a character string reporting the iOS device type. The ClientInfoIphone parameter may be one of the following values (note other device types may be added in future):

ClientInfoIphone	Description
i386	iPhone Simulator
iPhone1,1	iPhone
iPhone1,2	3G iPhone
iPhone2,1	3GS iPhone
iPhone3,1	4 iPhone
iPod1,1	1st Generation iPod
iPod2,1	2nd Generation iPod
iPod3,1	3rd Generation iPod

The following code within the form \$construct() could be used to return the current device type:

```
Calculate deviceType as pParams.ClientInfoIphone
```

Unique Device Identifier (UDID)

The pParams row variable contains the UDID parameter which is the unique ID of the current iOS device. The following code within the form \$construct() could be used to return the UDID:

```
Calculate deviceID as pParams.UDID
```

The UDID is useful for identifying individual devices or end users, allowing you to store some information in your app, such as the configuration of a Tabbar as selected by an end user, against individual UDIDs.

Component Transparency

All iOS components have the `$alpha` property which specifies the transparency of the component, with 0 being completely transparent and 255 being opaque; for most controls, `$alpha` is set to 255 by default. Some components also have the `$backalpha` property which controls the transparency of the background part of the control, rather than the foreground elements or data in the control.

Component Icons

Several of the iOS components allow you to use icons to enhance the user experience. This includes standard pushbuttons (iButton control), as well as the navbar, segmented, tabbar, and toolbar controls, where you can use an icon to represent a single button, segment, or tab. You can use an icon from any of the Omnis icon datafiles or the `#ICONS` system table in your library, although the latter is advised. The icon for a component or button is specified in its `$iconid` property. The icon used *must support Alpha values* to be displayed in iOS.

Alpha icons

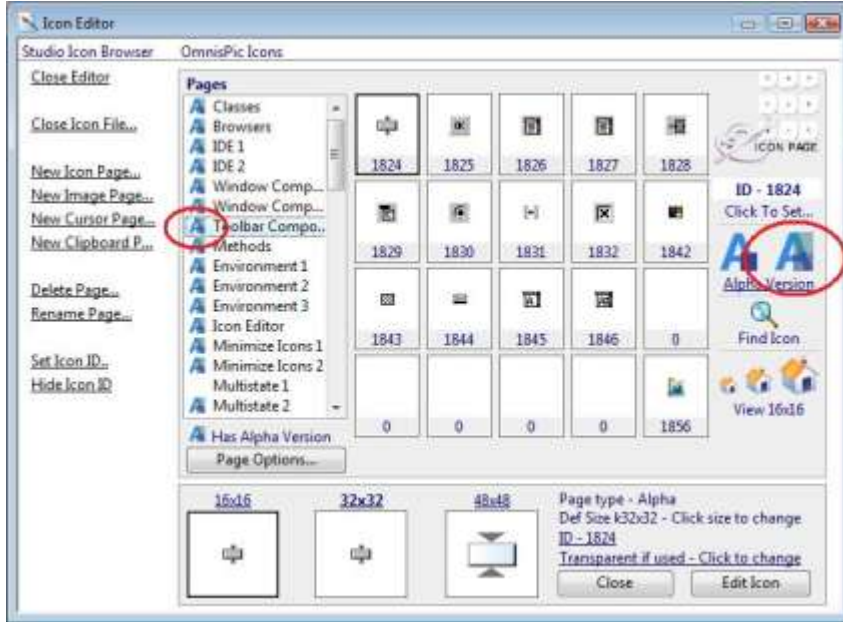
You must use icons with alpha values in components on iOS remote forms; non-Alpha icons are not supported for iOS components and will not display correctly. Therefore when you choose an icon for a component, by clicking on its `$iconid` property, you must choose an Alpha compatible icon from an Omnis icon datafile or the `#ICONS` system table in your library.

Icon pages

If you use an icon in any iOS component, such as a button or toolbar, you must specify the icon page name for the icon in the `$iconpages` property of the remote form. If the icon page is not listed in the `$iconpages` property, the icons will not be sent to the client and will not be displayed. So for example, if you have created your own icon page in `#ICONS`, you must check its name to the `$iconpages` property of the remote form.

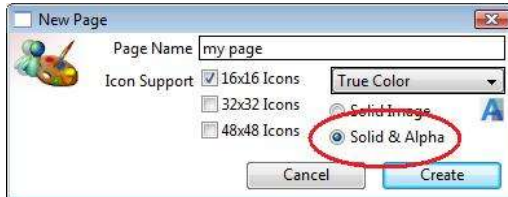
Creating your own icons

You can view or edit the icons in one of the Omnis icon datafiles, such as #ICONS, using the Omnis Icon Editor (open via the Tools>>Icon Editor option). You can add your own icons using the same tool, which is described in detail in the *Omnis Programming* manual in the *Library Tools* chapter.



To view the Alpha version of an icon page, you select the icon page and click on the large blue “A” icon on the right hand side on the Icon Editor window, as shown above. Pages that support Alpha icons are indicated in the Pages list using a small blue “A” icon, as shown above.

To create a new page, click on “New Icon Page” and make sure you select the “Solid & Alpha” option. In addition, you must check the boxes for 32x32 and 48x48 if you intend to add icons for this size.



Importing Images to an Icon file

When importing your own Alpha compatible icons into an Icon Page you must use the “Paste From File” option to import each Alpha image: you cannot use Copy and Paste from another image application since the Alpha properties may not be imported successfully using this method.



To import an Alpha icon, such as a PNG file, place your cursor in the icon field in the Icon Editor (as above), select Paste from File from the Edit menu, and select the image file you wish to import. Assign an ID to the icon and save the icon page. Note you can only add an icon of a particular size if the page supports that size: you can change the icon size support in the Page Options for a page.

#ICONS and icon pages

In most cases, you are advised not to edit or change the Omnis icon datafiles (Omnispic and Userpic), since many of the icons in these files are used in the Omnis environment itself. Rather you are advised to add your own icons to the #ICONS table in your library. You can open the #ICONS file by double-clicking it in the Studio Browser (it is in the System Classes folder in your library), or you can open it from within the Icon Editor.

You should also try to limit the number of icon pages used in your iOS app since all icon pages that contain icons in use (and listed \$iconpages) are sent to the client – if many icon pages are sent to the client, this may create an unnecessary burden on network traffic. Therefore you may want to add all the icons you need for your app to a single icon page in your #ICONS table – this will be more efficient and easier to maintain, especially if you are using the Omnis VCS to manage your library.

Styles

You can assign styles to iOS components based on the Field Styles defined in your library in the #STYLES system class. The definition for iOS styles are added under the kiOS section of the #STYLES system class.

Fonts

There is a new system class #IOSWFONTS to handle fonts on iOS devices. A new column called iOS has been added to the Window font table to allow you to map fonts used in desktop, web, and mobile forms, including iOS forms. (Note there is no equivalent font mapping for reports on mobile devices since Omnis reports do not run on mobile devices.)

Subforms, multiple forms and superclasses

You can add subforms to your remote forms allowing you, in effect, to embed one or more remote forms into a single remote form or application window. Using a single “main” form and a number of other forms loaded at runtime into a subform, would allow you to create a powerful and interactive iOS app.

The subform component is a standard Omnis component, but is available in the ‘iOS Components’ group in the Component Store. When you have placed the subform on your remote form, you specify the remote form to appear in the subform in its \$classname property. The subform field can be linked to a navbar or tabbar component using the

`$linkedobject` property, and depending on what the end-user clicks on, you can switch forms back and forth accordingly.

You can use multiple forms in the iOS app by switching from the initial form to another remote form using the `$changeform()` method.

You can use an empty remote form as the superclass of an iOS remote form and a non-iOS remote form simultaneously, which allows you to share code between iOS and non-iOS remote forms.

Paged panes

You can use a Paged pane in your iOS apps to simplify the user interface by placing a small number of fields and controls on separate panes and switching panes as appropriate.

Like the subform, the paged pane is a standard Omnis component and is available in the 'iOS Components' group in the Component Store. The paged pane can also be linked to a navbar or tabbar component via the `$linkedobject` property in the navigation component.

Setting form properties

When designing a remote form, you often need to click on the background of the form to set its properties in the Property Manager. This may be difficult if your form is completely filled with components and no form background is available to click on, as is often the case for mobile forms. To select the form in this case, you can use the Field List (right-click anywhere on the form, open the Field List and check the form name to open the Property Manager for the form), or if you click on any individual component, then shift-click it to deselect it, the focus will be returned to the form and its properties will be shown in the Property Manager.

iOS Preferences

The following commands are available for iOS remote forms only and allow you to save and load end-user data on the client, such as user preferences. To implement these preferences you need to use the `$clientcommand()` method which has the following general syntax:

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

savepreference

Saves a value (as a character string) as a named preference on the client. You could use this to store a username or a password for logging onto your app.

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

Where `row-variable` is `row(preference name, preference value)`.

loadpreference

Loads a named preference value from the client preferences into an instance variable.

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

Where `row-variable` is `row(preference name, instance variable name (e.g. a quoted string containing the name of the variable))`.

iOS Form Properties and Methods

Remote forms for iOS have all the properties and methods of a standard remote form together with a number of iOS specific properties, as well as some additional events. See the *Omnis Notation Reference* manual, or the Omnis Help (press F1), for a full description of the standard remote form properties and methods not listed here.

iOS Form Properties

Remote forms for iOS have many of the standard properties of a remote form together with a number of iOS specific properties described here.

\$alpha	The alpha value for the form, a value 0-255, where 0 is completely transparent, 255 is opaque
\$backalpha	The alpha component of the background color of the form, a value 0-255, where 0 is completely transparent, 255 is opaque
\$designshowmobiletitle	Hides or shows the title bar; if you deploy the form without the window title bar you need to disable the title bar in the client configuration; see below; in addition, if you hide the title, you'll need to add 20 pixels to the \$height of the form
\$designtaskname	You need to create a remote task and set this property to the remote task name; this is required to test the form
\$events	You have to enable specific events for the remote form, otherwise they will not be reported; see the Remote Form Events section below
\$iconpages	a comma separated list of icon pages that contain icons you have used for any components in your remote form; the icon pages listed here are sent to the client so it is usually more efficient to add all the icons you need for your iOS app to a single icon page, which must support Alpha (use the Omnis Icon Editor to add/edit icon pages); note icons/icon pages can be stored in #ICONS in the library
\$ios	Set this to kTrue to "switch" the remote form to an iOS compatible form; note this property will be grayed out if the remote form contains any components including standard web comps; the form has to be empty to set this property
\$screensize	Specifies the screen size and layout of the form; for iPhone and iPod devices this is kSSZiOS320x480Portrait kSSZiOS320x480Landscape for iPad this is kSSZiOS768x1024Portrait kSSZiOS768x1024Landscape Note you also need to set the \$height and \$width of the form for each size/orientation that you use in the form

iOS Form Methods

Remote forms for iOS have many of the standard methods of a remote form, as well as a number of iOS specific methods listed here. For example, `$showmessage()` can be used to display an OK message on the client.

Method	Description
<code>\$beginanimations()</code>	<code>\$beginanimations(iDuration [,iCurve=kiOSAnimationCurveEaseInOut, iRepeatCount=0, bAutoReverse=kFalse])</code> after calling this method, assignments to some properties are animated for <code>iDuration</code> milliseconds by <code>\$commitanimations()</code>
<code>\$commitanimations()</code>	<code>\$commitanimations()</code> animates the relevant property changes that have occurred after the matching call to <code>\$beginanimations()</code>
<code>\$redraw()</code>	<code>\$redraw([bSetcontents=kTrue, bRefresh=kFalse, bBobjs=kFalse])</code> redraws the window or field; <code>Do \$cinst.\$redraw()</code> redraws the form
<code>\$setcurfield()</code>	<code>\$setcurfield(vNameOrIdentOrItemref)</code> sets the current field on the client computer which is useful for data entry forms; when this is set the focus is placed in the field and on iOS the soft keypad is initiated; <code>\$setcurfield('')</code> removes the focus from the current field
<code>\$showmessage()</code>	<code>\$showmessage(cMessage [,cTitle])</code> Displays an OK message on the client computer using the specified <code>cMessage</code> and <code>cTitle</code>
<code>\$clientcommand()</code>	allows you to execute various functions on the iOS client, such as Yes/No messages and client preferences; see earlier in this chapter

Remote task methods

Together with the remote form methods, you can use the standard remote task methods in your iOS apps, including `$openform()` and `$changeform()`. For example, the following method can be placed behind a button to allow the end user to navigate back to the previous screen; in this case, the app appears to ‘remember’ the last page and uses the `$openform()` method to reopen the last form:

```
On evClick      ;; the last form visited is saved in tLastPage
    Do $openform(
        $ctask.tLastPage, kFormTransTypeFade, kFormTransDirNone)
```

Note the `$openform()` method in this case has three parameters: the name of the form to be opened, as well as the transition type and direction specified using one of the `kFormTransType..` and `kFormTransDir..` constants.

See the *Omnis Notation Reference* manual or the Omnis Help (press F1) for a full description of the remote task methods.

iOS Form Events

iOS enabled remote forms report the following events:

evAnimationsComplete	<p>The animation has completed</p> <p>Parameters</p> <table border="1"> <tr> <td data-bbox="682 340 829 387">pEventCode</td> <td data-bbox="829 340 1319 387">The event code</td> </tr> </table>	pEventCode	The event code		
pEventCode	The event code				
evFormToTop	<p>The remote form is about to become visible on the client</p> <p>Parameters</p> <table border="1"> <tr> <td data-bbox="682 473 829 520">pEventCode</td> <td data-bbox="829 473 1319 520">The event code</td> </tr> <tr> <td data-bbox="682 520 829 600">pScreenSize</td> <td data-bbox="829 520 1319 600">A kSSZ... constant for the current screen size on the client</td> </tr> </table>	pEventCode	The event code	pScreenSize	A kSSZ... constant for the current screen size on the client
pEventCode	The event code				
pScreenSize	A kSSZ... constant for the current screen size on the client				
evScreenOrientationChanged	<p>The orientation of the screen displaying the form has switched between portrait and landscape</p> <p>Parameters</p> <table border="1"> <tr> <td data-bbox="682 722 829 769">pEventCode</td> <td data-bbox="829 722 1319 769">The event code</td> </tr> <tr> <td data-bbox="682 769 829 848">pScreenSize</td> <td data-bbox="829 769 1319 848">A kSSZ... constant for the current screen size on the client</td> </tr> </table>	pEventCode	The event code	pScreenSize	A kSSZ... constant for the current screen size on the client
pEventCode	The event code				
pScreenSize	A kSSZ... constant for the current screen size on the client				
evSubFormToTop	<p>An existing remote form, contained in a subform that has \$multipleclasses set to kTrue, is about to become visible on the client</p> <p>Parameters</p> <table border="1"> <tr> <td data-bbox="682 999 829 1046">pEventCode</td> <td data-bbox="829 999 1319 1046">The event code</td> </tr> </table>	pEventCode	The event code		
pEventCode	The event code				

Note Context menu events (evExecuteContextMenu and evOpenContextMenu) are not available to iOS enabled forms.

Events for iOS Components

The new iOS components have events which can be handled in the same way as with previous Omnis applications using the On <event> command in the \$event() method of the component. However the event handling methods cannot be run in the client and must therefore be executed on the Omnis Server. Events for each component are listed in their respective section in the *iOS Components* section below.

Event Handling Methods

If you double-click on a component or the form background, the method editor will open showing the event handling method for the component or form. In most cases, the On <event> command is added to the method, allowing you to add your own code to handle the event or multiple events.

Enabling Events

You should remember to enable any events that you wish to report for any individual component or the remote form itself by enabling the event in the \$events property for the component or form. If an event is not enabled in the \$events property, it will not be reported, even if you have added code to handle the event in the \$event method of the component or form.

To enable an event, select the component or form background, open the Property Manager (press F6), and click on the \$events property to open the event droplist. Select the event you wish to report from the list of possible events for the component or form.

iOS Components

There are several new components for remote forms that you can use on iOS devices, and many of them will be familiar to iPhone/iPad users. When the components are instantiated on the device itself, the native iOS components are used. The following section lists all the iOS specific components available for remote forms. In addition, you can use the Paged Pane and Subform components for iOS forms.

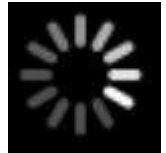
Example Omnis iOS app

This release includes an Omnis iOS app that allows you to play an Anagram based word game. The example app shows the features of some of the new iOS components, including how you write Omnis code to control how they behave. Some of the following sections use code snippets from the example application, which you can further examine in Omnis itself looking at the comments to navigate around the app.



iActivity Control

iActivity provides an animated image to show some activity on the client, for example, during a long list calculation or search operation. You assign kTrue to the \$animating property to display the animated image. The \$hidewhensstopped property controls if the control is hidden when animation stops.



The following method can be used to display the activity component, which could be called or triggered by some event in your application.

```
Do $cinst.$objs.oActivity.$animating.$assign(kTrue)
```

Events

The iActivity control has no events.

iButton Control

The Button field responds to user clicks reported as the evClick event which can be handled in the \$event() method of the button. The button can display an icon, specified in the \$iconid property, and/or a single line of text. If you use an icon for the button, you must specify the icon page for the icon in the \$iconpages property of the remote form. In addition, the icon must support Alpha.

Events

evClick	a user generated click
---------	------------------------

The following method can be placed behind a button control, which, in this case, allows the user to switch to another form.



```
; code behind Scores button
```

On evClick

```
Calculate tLastPage as 'rfTitle' ;; Store this page so we can  
come back here from the scores page
```

```
Do $ctask.$openform(  
    'rfScoreboard', kFormTransTypeFlipLeft, kFormTransDirFromLeft)
```

iDateTime Control

The iDateTime component provides a “spinner” field to allow the end user to select dates and/or times. You can assign a Date/Time instance variable to the \$dataname property of the control to load the date/time selected by the user. The \$pickerstyle property specifies the style of the date time picker. The following image shows the default Date/Time picker style showing the current date.



Events

evClick	a user generated click
---------	------------------------

You can create an instance variable in your form called iDate with the Subtype ‘Date Time D m y’ and assign the variable to the \$dataname property of the Datetime control. In the

\$construct() method of your form you can use the following method to assign today's date to the variable.

```
Calculate iDate as #D
```

When the form is opened the Datetime control will display today's date. When the user selects a different date, the selected date will be held in the iDate variable. For example, you could ask the end user to select their date of birth and the following code would calculate the end user's age this year.

```
Calculate lAge as ddiff(kYear,iDate,#D)
```

```
Do $cinst.$showmessage(con('You are ',lAge,' years old this year.'))
```

imageView Control

The imageView comp can be used to display an image, either a TIFF, JPEG, GIF, PNG, BMP, ICO, CUR, or XBM image. The control supports the drag and pinch gestures which allow the end user to zoom into and move the image within the picture control.

The \$dataname of the control must be set to a Binary variable containing the image data; you cannot use an Omnis picture variable. Alternatively, the variable in \$dataname can be an integer which references an icon id to display an icon from one of the Omnis icon datafiles or #ICONS in your library.

The imageView control has the following properties:

Property	Description
\$getimagedata	(runtime only) assign a kiImageSource... constant to this property to instruct the client to prompt the user for an image from the specified source, and send an evImageSelected event with the image data in pImageData
\$getrawimage	(runtime only) assign a kiImageSource... constant to this property to instruct the client to prompt the user for an image from the specified source, and use the selected image to set \$rawimage
\$imagealign	specifies where the image will be positioned when \$noscale is kTrue and the image entirely fits in the bounds of the control
\$maxzoom	The maximum zoom factor that can be applied to the current image by the pinch gesture (1-16)
\$rawimage	if set, this raw PNG image is used instead of the databound object

Events

evImageSelected	Sent to the control when the action requested by assigning \$getimagedata completes; the pImageData parameter contains the binary image data selected by the user
-----------------	---

The following method can be placed behind a toolbar control that allows the end user to select images from the photo or camera library on their device. The \$getimagedata property must be assigned at runtime and can be used with either kiImageSourcePhotoLibrary or kiImageSourceCameraRoll to specify the image source.

```

; oPic is the image control, its dataname is
  iPic which is a binary variable
On evClick
  Switch pToolbarButton ;; contains the tab
    number clicked
  Case 1
    Do $cinst.$objs.oPic.$getimagedata.$assign(
      kiImageSourcePhotoLibrary) Returns #F
  Case 2
    Do $cinst.$objs.oPic.$getimagedata.$assign(
      kiImageSourceCameraRoll) Returns #F
  End Switch
  Do $cinst.$redraw()

```



The event method behind the image control loads the image held in pImageData and assigns it to the iPic binary variable behind the image control.

```

On evImageSelected
  Calculate iPic as pImageData
  Do $cinst.$redraw()

```

iLabel Control

The iLabel control provides a simple text label.

Property	Description
\$adjustfontsizefitwidth	if true, and \$numberoflines is one, the object reduces the font size in order to fit the text string into its bounding rectangle - the property \$minimumfontsize specifies the smallest value to which the font size can be reduced
\$linebreakmode	a kiOSLineBreakMode... constant that specifies how lines break when drawing text
\$minimumfontsize	the smallest value to which the font size can be reduced when \$adjustfontsizefitwidth is true and \$numberoflines is one
\$numberoflines	the number of lines to be used to render the text. Zero means use as many as necessary

Events

The iLabel control has no events.

iMap Control

The iMap control provides a Google map interface to allow the end user to search for locations or for you to show locations in your app. The end user can drag and pinch the map image to change location and zoom of the map. Using the map control together with other controls, such as the table control which could list a number of locations, you can create very interactive location based Omnis apps.

You can display the map with a specified location, adding your own annotation pins and/or popup location markers. The iMap control generates a number of events in response to the end user touching the map or markers.

The iMap control has the following properties:

Property	Description
\$addannotation	Adds pin annotations to the map. Assign a list with columns (nLatitude, nLongitude, cTitle [,cDesc]). nLatitude and nLongitude are degrees where positive numbers are North and East, or negative numbers are South and West. cTitle is the title for the marker with cDesc the optional description
\$centermap	Centers the map to the location. Assign a row with columns (nLatitude, nLongitude, nSpan). nLatitude and nLongitude are degrees where positive numbers are North and East or negative numbers are South and West. nSpan is the number of degrees of latitude and longitude to display
\$desiredaccuracy	A kiMapLocationAccuracy... constant which indicates the level of accuracy you require for evHeadingChanged and evLocationChanged events. Greater accuracy requires more time and power
\$enableheadingevents	If true, heading events are sent from the device to the server (the device must support heading events)
\$enablelocationevents	If true, location events are sent from the device to the server (the device must support location events)
\$headingfilter	The minimum angular change (measured in degrees) required to generate new heading events. Set this to zero to indicate no filtering of evHeadingChanged events
\$locationfilter	The minimum distance (measured in meters) the device must move laterally before another evLocationChanged event is generated. Set this to zero to indicate no filtering of evLocationChanged events
\$mapcanscroll	If true, the map can scroll
\$mapcanzoom	If true, the map can zoom
\$maptype	The type of map, a constant: kiMapTypeStandard, kiMapTypeSatellite or kiMapTypeHybrid

Property	Description
\$nexttouchevent	If kTrue, the next touch on the map sends an evLocationTouch event. pLongitude and pLatitude are sent as parameters
\$showannotation	Assign an annotation number (a 1-based index into the added annotations) to show the callout bubble for the annotation; the number usually relates to the order of the list you may have assigned to \$addannotation. If the annotation is not currently visible, assigning this property has no effect
\$showuserlocation	If true, the map shows the users location

Events

The iMap control generates a number of events which you can detect in your event handling methods for the control. Here is a summary of the events; the next section provides full details and parameters for each event.

- evAnnotationTouched**
Sent to the control when an annotation is touched
- evLocationChanged**
Sent to the control when the location changes or when \$enablelocationevents changes value to kTrue. You can use \$locationfilter to restrict the generation of evLocationChanged events
- evHeadingChanged**
Sent to the control when the heading changes or when \$enableheadingevents changes value to kTrue. You can use \$headingfilter to restrict the generation of evHeadingChanged events
- evLocationOrHeadingError**
Sent to the control when an error occurs while trying to get location or heading data
- evLocationTouch**
Sent to the control when \$nexttouchevent is kTrue. After the event \$nexttouchevent will be set to kFalse

Event	Description and parameters																			
evAnnotationTouched	<p>Sent to the control when an annotation is touched</p> <p>Parameters</p> <table border="1" data-bbox="605 314 1279 413"> <tr> <td data-bbox="605 314 805 366">pEventCode</td> <td data-bbox="808 314 1279 366">The event code</td> </tr> <tr> <td data-bbox="605 369 805 413">pAnnotationRow</td> <td data-bbox="808 369 1279 413">The row number of the annotation touched</td> </tr> </table>		pEventCode	The event code	pAnnotationRow	The row number of the annotation touched														
pEventCode	The event code																			
pAnnotationRow	The row number of the annotation touched																			
evLocationChanged	<p>Sent to the control when the location changes or when \$enablelocationevents changes value to kTrue. You can use \$locationfilter to restrict the generation of evLocationChanged events</p> <p>Parameters</p> <table border="1" data-bbox="605 612 1316 1515"> <tr> <td data-bbox="605 612 851 664">pEventCode</td> <td data-bbox="853 612 1316 664">The event code</td> </tr> <tr> <td data-bbox="605 668 851 743">pLatitude</td> <td data-bbox="853 668 1316 743">The latitude in degrees (positive for North or negative for South)</td> </tr> <tr> <td data-bbox="605 746 851 821">pLongitude</td> <td data-bbox="853 746 1316 821">The longitude in degrees (positive for East or negative for West)</td> </tr> <tr> <td data-bbox="605 824 851 876">pAltitude</td> <td data-bbox="853 824 1316 876">The altitude in meters</td> </tr> <tr> <td data-bbox="605 880 851 954">pHorizontalAccuracy</td> <td data-bbox="853 880 1316 954">The radius of uncertainty for the location, measured in meters</td> </tr> <tr> <td data-bbox="605 958 851 1154">pVerticalAccuracy</td> <td data-bbox="853 958 1316 1154">The accuracy of the altitude value in meters. The value in the altitude property could be plus or minus the value indicated by this property. A negative value indicates that the altitude value is invalid</td> </tr> <tr> <td data-bbox="605 1157 851 1232">pSpeed</td> <td data-bbox="853 1157 1316 1232">The instantaneous speed of the device in meters per second</td> </tr> <tr> <td data-bbox="605 1236 851 1432">pCourse</td> <td data-bbox="853 1236 1316 1432">The direction in which the device is travelling. Course values are measured in degrees starting at zero for due north and continuing clockwise around the compass. A negative value indicates that the course value is invalid</td> </tr> <tr> <td data-bbox="605 1435 851 1515">pDescription</td> <td data-bbox="853 1435 1316 1515">A text string representing the location or heading data</td> </tr> </table>		pEventCode	The event code	pLatitude	The latitude in degrees (positive for North or negative for South)	pLongitude	The longitude in degrees (positive for East or negative for West)	pAltitude	The altitude in meters	pHorizontalAccuracy	The radius of uncertainty for the location, measured in meters	pVerticalAccuracy	The accuracy of the altitude value in meters. The value in the altitude property could be plus or minus the value indicated by this property. A negative value indicates that the altitude value is invalid	pSpeed	The instantaneous speed of the device in meters per second	pCourse	The direction in which the device is travelling. Course values are measured in degrees starting at zero for due north and continuing clockwise around the compass. A negative value indicates that the course value is invalid	pDescription	A text string representing the location or heading data
pEventCode	The event code																			
pLatitude	The latitude in degrees (positive for North or negative for South)																			
pLongitude	The longitude in degrees (positive for East or negative for West)																			
pAltitude	The altitude in meters																			
pHorizontalAccuracy	The radius of uncertainty for the location, measured in meters																			
pVerticalAccuracy	The accuracy of the altitude value in meters. The value in the altitude property could be plus or minus the value indicated by this property. A negative value indicates that the altitude value is invalid																			
pSpeed	The instantaneous speed of the device in meters per second																			
pCourse	The direction in which the device is travelling. Course values are measured in degrees starting at zero for due north and continuing clockwise around the compass. A negative value indicates that the course value is invalid																			
pDescription	A text string representing the location or heading data																			

Event	Description and parameters										
evHeadingChanged	<p>Sent to the control when the heading changes or when \$enableheadingevents changes value to kTrue. You can use \$headingfilter to restrict the generation of evHeadingChanged events</p> <p>Parameters</p> <table border="1" data-bbox="605 383 1310 847"> <tr> <td data-bbox="605 383 829 428">pEventCode</td> <td data-bbox="832 383 1310 428">The event code</td> </tr> <tr> <td data-bbox="605 432 829 512">pMagneticHeading</td> <td data-bbox="832 432 1310 512">The heading (measured in degrees) relative to magnetic North</td> </tr> <tr> <td data-bbox="605 515 829 595">pTrueHeading</td> <td data-bbox="832 515 1310 595">The heading (measured in degrees) relative to true North</td> </tr> <tr> <td data-bbox="605 598 829 765">pHeadingAccuracy</td> <td data-bbox="832 598 1310 765">The maximum deviation (measured in degrees) between the reported heading and the true geomagnetic heading. A negative value means that the reported heading is invalid</td> </tr> <tr> <td data-bbox="605 769 829 847">pDescription</td> <td data-bbox="832 769 1310 847">A text string representing the location or heading data</td> </tr> </table>	pEventCode	The event code	pMagneticHeading	The heading (measured in degrees) relative to magnetic North	pTrueHeading	The heading (measured in degrees) relative to true North	pHeadingAccuracy	The maximum deviation (measured in degrees) between the reported heading and the true geomagnetic heading. A negative value means that the reported heading is invalid	pDescription	A text string representing the location or heading data
pEventCode	The event code										
pMagneticHeading	The heading (measured in degrees) relative to magnetic North										
pTrueHeading	The heading (measured in degrees) relative to true North										
pHeadingAccuracy	The maximum deviation (measured in degrees) between the reported heading and the true geomagnetic heading. A negative value means that the reported heading is invalid										
pDescription	A text string representing the location or heading data										
evLocationOrHeadingError	<p>Sent to the control when an error occurs while trying to get location or heading data</p> <p>Parameters</p> <table border="1" data-bbox="605 961 1310 1328"> <tr> <td data-bbox="605 961 829 1006">pEventCode</td> <td data-bbox="832 961 1310 1006">The event code</td> </tr> <tr> <td data-bbox="605 1010 829 1216">pSystemErrorCode</td> <td data-bbox="832 1010 1310 1216">The system error code for the problem encountered when getting location or heading data. If the user has denied access to the information, the control sets \$enablelocationevents and \$enableheadingevents to kFalse</td> </tr> <tr> <td data-bbox="605 1220 829 1328">pSystemErrorText</td> <td data-bbox="832 1220 1310 1328">The system error text for the problem encountered when getting location or heading data</td> </tr> </table>	pEventCode	The event code	pSystemErrorCode	The system error code for the problem encountered when getting location or heading data. If the user has denied access to the information, the control sets \$enablelocationevents and \$enableheadingevents to kFalse	pSystemErrorText	The system error text for the problem encountered when getting location or heading data				
pEventCode	The event code										
pSystemErrorCode	The system error code for the problem encountered when getting location or heading data. If the user has denied access to the information, the control sets \$enablelocationevents and \$enableheadingevents to kFalse										
pSystemErrorText	The system error text for the problem encountered when getting location or heading data										
evLocationTouch	<p>Sent to the control when \$nexttouchevent is kTrue. After the event \$nexttouchevent will be set to kFalse</p> <p>Parameters</p> <table border="1" data-bbox="605 1442 1310 1484"> <tr> <td data-bbox="605 1442 758 1484">pEventCode</td> <td data-bbox="761 1442 1310 1484">The event code</td> </tr> </table>	pEventCode	The event code								
pEventCode	The event code										

Event	Description and parameters	
	pLatitude	The latitude in degrees (positive for North or negative for South)
	pLongitude	The longitude in degrees (positive for East or negative for West)

iMap example

Consider an example app that contains an iMap control and a table containing a number of locations, plus a toolbar for adding markers. The contents of the table list could be built on the fly or from a database, which is the case in this example. The following code samples show how you can build the location list and show each location with an annotation pin.

The first task would be to create a database session, logon to the database and build the list of locations. In this case the locations are held in an Omnis database located in the same folder as the app, and standard SQL code is used to build the list based on the Markers table.

```

; $getAnnotation class method
; Instance var: iAnnList (List)
; oMap is the map control on the form
Do iAnnList.$clear()
Do $cinst.$objs.oMap.$addannotation.$assign('')
Do iStmt.$execdirect('SELECT * FROM Markers') Returns #F
Do iStmt.$fetch(iAnnList, kFetchAll)
Do $cinst.$objs.oMap.$addannotation.$assign(iAnnList)
Do $cinst.$redraw()
    
```

In this example, the iAnnList list variable has the following data.



The screenshot shows the 'Variable Inspector' window for the variable 'iAnnList'. It displays a table with the following data:

	Latitude	Longitude	Title	Description	Flag
1	52.198677396	1.47826194763	Mitford House	We are here!	2
2	53.6374122866	10.0151324272	Tigerlogic Germany	Tigerlogic's German office	2
3	33.6507438877	-117.737216949	Tigerlogic Corporation	Tigerlogic HQ	2
4	48.8522631676	2.39733695984	Tigerlogic France	Tigerlogic's French office	2

At the bottom of the window, there are two tabs: 'iAnnList' and 'List'.

The list of annotation data is assigned to the map control by assigning it to its \$addannotation property. The data list must contain columns for the Latitude and Longitude of the locations, a Title for the location marker, and an optional description. In our example there is a fourth column containing the type of accessory marker to be displayed in the location list on the form. The Latitude and Longitude parameters are specified as the number of degrees where positive numbers are North and East, and negative numbers are South and West.

In this example, the remote form startup code centers the map and displays the second pane of a paged pane control showing the table of locations. The instance variable iCurrentPos is a Row based on the schema table in our database (set the subtype of the variable to the name of the schema class), with the columns Latitude, Longitude, and Span. The value of iCurrentPos is hard coded in this case and assigned to the \$centermap property of the map control.

```
Do iCurrentPos.$assigncols(52.3,1.65,4)
Do $cinst.$objs.oMap.$centermap.$assign(iCurrentPos)
Calculate $cinst.$objs.oPane.$currentpage as 2
Do $cinst.$redraw()
```

In this example, event handling code has been placed behind the table control, so when the end user taps the list or the accessory icon either the corresponding marker is displayed or the map is centered on the chosen location. Here is the code behind the table control.

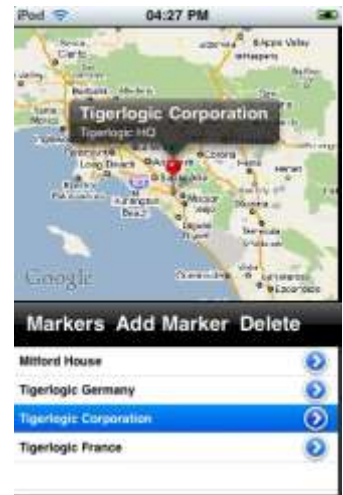
```
On evTableRowAccessoryClicked
    Do iCurrentPos.$assigncols(
        iAnnList.[pRow].Latitude,
        iAnnList.[pRow].Longitude,
        iSpan)
    Do $cinst.$objs.oMap.$centermap.$assign(
        iCurrentPos)
    Do $cinst.$redraw()
```

The evTableRowAccessoryClicked event returns the row number of the Accessory icon the user has tapped in pRow which can be used to get the Latitude and Longitude from the list of annotations in iAnnList. The selected location in iCurrentPos is then assigned to the \$centermap property and the map is centered automatically on the location.

The other event handling method behind the table is called when the end user taps a line in the table.

```
On evTableRowClicked
    Do $cinst.$objs.oMap.$showannotation.$assign(pRow)
```

The evTableRowClicked event returns the row number the user has tapped in pRow which can be assigned directly to the \$showannotation property of the map control.



You can add location markers to the map using the `evLocationTouch` event to detect where the end user has tapped and adding the location returned in the event to the list of annotations. In our example, a button is used to enable the touch event for the map control and two fields are provided to allow the end user to enter a name and description. Here is the method behind the 'Place Marker' button:

```
On evClick
    Calculate $cinst.$objs.oMap.$nexttouchevent as kTrue
    Calculate $cinst.$objs.oPane.$objs.bCancel.$enabled as kTrue
    Calculate $cobj.$enabled as kFalse
    Do $cinst.$setcurfield('eName')
```

The method enables the touch event, enables the cancel buttons, disables itself, and sets the edit focus in the name field. Doing the latter will open the soft keyboard prompting the end user to enter a name. When the end user has entered a name and description for the marker, they can tap the device which will now trigger the `evLocationTouch` event. The event handling method is placed behind the map control and has the following code:

```
On evLocationTouch
    Do iAnnList.$add(
        pLatitude,pLongitude,iName,
        iDesc,kiTableCellAccessoryDisclosureButton)
    Do iStmt.$execdirect('INSERT INTO Markers
        VALUES (@[pLatitude],[pLongitude],[iName],[iDesc],2)')
    Returns #F
    Do $cinst.$getAnnotation() ;; builds the annotation list
```

The `evLocationTouch` event returns the selected location in the `pLatitude` and `pLongitude` parameters which can be used to add the location to the list and insert the new location into the database. The `$getAnnotation()` class method is called to rebuild the annotation list and redraw the form including the new location marker.

iMultiLineEdit Control

The `iMultiLineEdit` control is a field for displaying multiple lines of text from the variable specified in `$dataname`. The `$autocapital` property controls the capitalization of entered text, while `$autocorrect` specifies if auto correction is used.

See the section on the *SingleLineEdit Control* for further information about using edit fields in remote forms.

Events

evAfter	Sent to a field when it ceases to be the target field
evBefore	Sent to a field when it becomes the target field

iNavigationbar Control

The iNavigationbar component provides a standard iOS navigation bar which end users can use to navigate to different parts of your application. The navigation bar has a main title in the middle of the control and can have a left and/or right button which respond to user clicks.

Property	Description
\$initiallefticonid	if this is not zero, and \$initiallefttext is empty, the first navigation bar item has a button on the left hand side, displaying this icon
\$initiallefttext	if this is not empty, the first navigation bar item has a button on the left hand side, displaying this text
\$initiallefttype	specifies the type of button displayed on the left hand side of the first navigation bar item
\$initialrighticonid	if set to kinavigationbarbuttontypeimage, the first navigation bar item has a button on the right hand side, displaying this icon
\$initialrighttext	if set to kinavigationbarbuttontypetext, the first navigation bar item has a button on the right hand side, displaying this text
\$initialrighttype	specifies the type of button displayed on the right hand side of the first navigation bar item
\$initialtitle	is the initial title displayed on the navigation bar
\$lefthidden	if true, the left hand (back) button is hidden for the current navigation bar stack item
\$linkedobject	the name of a subform or paged pane object on the current remote form, used in conjunction with the \$push property of the navbar. If you use a subform, \$multipleclasses for the subform must be ktrue
\$navigationbarstyle	specifies the appearance of the navigation bar
\$push	allows you to assign a 2-4 column row to the object referenced in \$linkedobject, col1 is the page number of a paged pane or classname of the linked subform, col2 is the title for pushed item, col3 is the text or icon id for right button (pass empty for no right button), and col4 can be non-zero to hide the left button
\$righticonid	if this is not zero, and \$righttext is empty, the current navigation bar item has a button on the right hand side, displaying this icon
\$righttext	if this is not empty, the current navigation bar item has a button on the right hand side, displaying this text
\$righttype	specifies the type of button displayed on the right hand side of the current navigation bar item
\$tintColor	the color of the navigation bar
\$title	the title for the current navigation bar stack item

If you use icons in your navbar you must specify the icon page for the icon(s) in the \$iconpages property of the remote form.

Events

evClickInitialLeftButton	The initial Left Button has been clicked
evClickRightButton	The Right Button has been clicked

Example Navigation bar



Consider the Help form in the Anagrams example application. The Help form has a navbar which allows the end user to select different help topics. The navbar itself is linked to a paged pane field which displays the help topics on individual panes. The \$linkedobject property of the navbar specifies the name of the paged pane, in this case, called oPane.

The \$construct() method of the Help form builds a list containing information about the pages of the tab pane, including the arguments needed for the \$push property of the navbar.

```
Do iPageList.$define(Page,Title,Rightbtn,HideLeft)
Do iPageList.$add(1,'Info','Manual Play',1)
Do iPageList.$add(2,'Manual Play','Anagram',0)
Do iPageList.$add(3,'Anagram','',0)
```

The navbar control itself is placed across the top of the form and its various properties under the General and Appearance tabs in the Property Manager are set, as follows:

\$events	set to receive evClickRightButton events
\$linkedobject	set to oPane, the name of the paged pane
\$push	can only be assigned at runtime; see below
\$initialtitle	set to "Main"
\$initialrighttype	set to kiNavigationbarButtonTypeText
\$initialrighttext	set to "Manual Play"

The \$event() method traps a user click on the button on the navbar, and has the following event code:

```
On evClickRightButton
    Do $cinst.$pushPage(iPage+1)    ;; page number is incremented
    and passed to the $pushPage method
```

The \$pushPage method is a class method and gets the details for the new page from iPageList (built in the \$construct of the form, as above) and passes the details to the \$push property of the navbar.

```

; pPage receives the page number
; lRow is local var of Row type
; iPage stores the current page number
Calculate lRow as iPageList.[pPage]
Do $cinst.$objs.oNav.$push.$assign(lRow)
Calculate iPage as pPage
    
```

The effect of the \$push property is to change the pane number in the paged pane control specified in the \$linkedobject property; in the case of the example app, an initial click by the end user will display the second pane in the Help form.

iPage Control

The iPage control links to a Paged pane on the remote form and allows the end user to change the current page in the linked paged pane by flicking or swiping over the page control. The paged pane to link to the iPage control is specified in the \$linkedobject property.



The iPage control also gives the end user a visual clue as to the current selected pane in the linked page pane object, since the highlighted dot in the control changes to reflect the current page in the linked paged pane (the screenshot shows page 3 selected).

Property	Description
\$currentpage	the current page number
\$linkedobject	the name of a paged pane object on the current remote form that links to the iPage control
\$pagecount	the number of pages

Events

evPageChanged	The page has changed; pValue = the new page
---------------	--

Paged Pane Control

The standard Omnis paged pane can be used in iOS-enabled remote forms. The paged pane provides a very convenient method to show alternative fields or controls, or to break down an entry form into more manageable parts whereby each pane contains a small number of fields.

Several of the other iOS controls can link to a paged pane, by setting the control's \$linkedobject property to the name of the paged pane. You can link a paged pane to the iNavbar, iPage, and iTabbar controls.

In addition to the standard paged pane properties, you can set \$effect to select different border effects for the control, and by setting \$scrolltochange to kTrue the end user is able to change the current page simply by flicking or wiping horizontally across the paged pane.

iProgress Control

The iProgress control lets you indicate the progress of an operation such as a complex search, calculation, or loop. The \$progressstyle sets the style of the progress bar. The \$min and \$max properties specify the maximum and minimum values for the progress range, while \$val is the current value in the progress range (between \$min and \$max).



Events

evClick	a user generated click
---------	------------------------

The following method could be placed behind a button to trigger the progress control, but could equally be triggered by another event in your code.

```
; the $min and $max of 'progress' are set to 0 and 1000
On evClick
  For count from 1 to 1000 step 1
    Do $cinst.$objs.progress.$val.$assign(count)
  End For
  Do $cinst.$showmessage('Done!')
```

iSearchBar Control

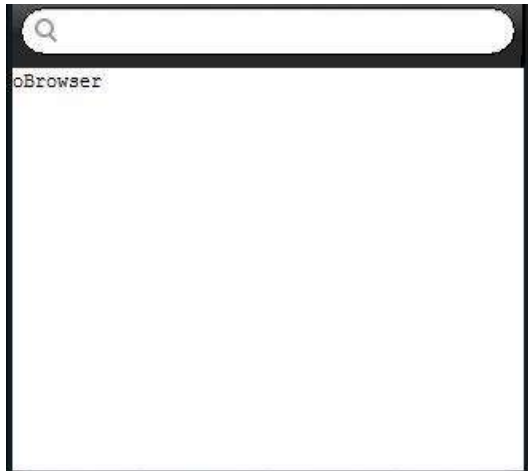
The iSearchBar component provides an entry field in the search bar style. The text entered by the user is held in the variable specified in \$dataname and can be used as the source for a search.

Property	Description
\$autocapital	controls the capitalization of text entered
\$autocorrect	controls how the iOS device corrects the entered text
\$contenttip	the text displayed in the field when it is empty, to help the user understand what content should be entered
\$dataname	the data name of the object
\$prompt	the text shown above search bar; increase the height of the searchbar control itself to make the text visible
\$searchbarstyle	sets the visual style of the searchbar (only applies if \$tintcolor is kColorDefault)
\$showsbookmark	enables the bookmark icon in the search bar; note you can detect a click on the bookmark with evBookmarkClick which must be enabled in the \$events property
\$showscancel	enables the cancel button in the search bar
\$tintcolor	the color of the search bar

Events

evAfter	Sent to a field when it ceases to be the target field
evBefore	Sent to a field when it becomes the target field
evBookmarkClick	The user has pressed the bookmark button
evClick	a user generated click

The following Searchbar control works in conjunction with a Webview control to allow the end user to browse web pages within your Omnis application. The \$dataname of the searchbar is set to iUrl, a simple Character instance variable.



The code in the \$event() method for the Searchbar assigns the contents of the control held in iUrl, along with a Webview command constant, to a row variable which is itself assigned to the \$execcommand property of the Webview control.

```
On evClick
    Do lCommand.$define('','')
    Calculate lCommand.1 as kiWebViewCommandLoadPage
    Calculate lCommand.2 as iUrl ;; the text in the Searchbar
    Do $cinst.$objs.oBrowser.$execcommand.$assign(lCommand)
    Do $cinst.$setcurfield('')
    Do $cinst.$redraw()
```

iSegmented Control

The iSegmented component is similar to the tabbar in so far as it provides a number of buttons or segments for the user to click. You can detect which segment the user has clicked on and execute the appropriate code.

Property	Description
\$currentsegment	the number (1 - \$segmentcount) of the current segment (this specifies the segment affected by segment specific properties);
\$movesegment	moves the segment by assigning a number in the range 1 to \$segmentcount, which changes \$currentsegment to the assigned number (not assignable in class notation)
\$segmentcount	the number of segments (must be at least one);
\$segmentenabled	if true, the segment is enabled and generates a click event when the user presses it;
\$segmenticonid	the icon displayed on the current segment, if \$segmenttext is not empty, \$segmenticonid is ignored;
\$segmentlist	a list containing segment-specific properties, one line per segment;
\$segmentstyle	specifies the appearance of the control;
\$segmenttext	the text displayed on the current segment, if this is not empty, \$segmenticonid is ignored;
\$segmentwidth	the width of the segment in pixels, otherwise if zero, the control automatically sizes the segment;
\$selectedsegment	the number (1 - \$segmentcount) of the currently selected segment, or zero if no segment is selected (only relevant if \$showselectedsegment is kTrue);
\$showselectedsegment	if true, the control highlights the selected segment (\$selectedsegment specifies the highlighted segment);
\$tintcolor	the color of the control (only applies if \$segmentstyle is kiSegmentStyleBar and \$tintcolor is not kColorDefault);

If you use icons in the segmented control you must specify the icon page for the icon(s) in the \$iconpages property of the remote form.

Events

evClick	a user generated click; pSegment = the number of the segment clicked
---------	---

The following Segmented control has been configured to work in conjunction with a Webview control to provide a web browser in a remote form.



The Segmented control has the following properties set:

\$event	evClick is enabled
\$segmentstyle	set to kiSegmentStyleBar
\$selectedsegment	set to 0 for no selection
\$segmentcount	is 4
\$currentsegment	you have to set this to 1 to 4 to assign an icon or text to each segment; in the above example, icons 3002, 3003, 1614, and text 'HOME' are used respectively

See the iWebView section for the example code used behind such a Segmented control.

iSingleLineEdit Control

The SingleLine Edit Field is for displaying or receiving a single line of text from the variable specified in \$dataname. In most cases the variable will be character based, but the single-line edit can display numeric data; see below. Like all remote form fields, the variable specified in \$dataname for the edit field must be an instance variable.

Property	Description
\$adjustsfontsizetofitwidth	if true, the object reduces the font size in order to fit the text string into its bounding rectangle, while \$minimumfontsize is the smallest value to which the font size can be reduced when \$adjustsfontsizetofitwidth is true
\$autocapital	controls the capitalization of text entered
\$autocorrect	controls how the iOS device corrects the entered text
\$clearbuttonmode	controls when the clear button is displayed in the field which allows the end user to clear any data in the field; this is one of the kiOSViewMode... constants
\$contenttip	the text displayed in the field when it is empty, to help the user understand what content should be entered
\$effect	The effect or border style of the edit field; for iOS this is one of the kiOSBorder... constants
\$dataname	the data name of the object
\$securetextentry	if true, each character is hidden which is useful for password text entry

Events

Single line edit fields do not report any special events other than the standard `evAfter`, `evBefore`, and `evClick`.

<code>evAfter</code>	Sent to a field when it ceases to be the target field
<code>evBefore</code>	Sent to a field when it becomes the target field
<code>evClick</code>	a user generated click

Using Edit fields with the keyboard

When a remote form containing edit fields is opened on the client, the edit focus is placed in the first available edit field and the built-in alpha-numeric keyboard is opened automatically ready for data entry. The end user can use all the editing features they would expect on an iOS device within an Omnis entry field, such as clicking and/or dragging to select text, copying & pasting text, predictive text, and so on.

For a single line edit field, you can set the `$clearbuttonmode` property to `kiOSViewModeWhileEditing` to enable the Cancel button during editing to allow the end user to clear the contents of the field (as shown).

Setting the edit focus

The edit focus is placed in the first available edit field according to the values of the `$order` property for each field in the form. You can specify that a field is the first edit field by changing its `$order` property to 1; note the `$order` values of the other fields in the form are reordered accordingly when you change the `$order` for a field. Alternatively, you can use the `$setcurfield()` method to specify which edit field gets the focus; this can be done in the `$construct()` of the form to put the edit focus in a particular field when the form is opened. For example, the following method will put the edit focus in the `FirstName` field and open the keyboard for data entry when the form is opened.

```
; $construct of the form
; $cinst is the remote form instance
Do $cinst.$setcurfield('FirstName')
```

You can cancel the edit focus, and by doing so close the soft keyboard, by setting the current field in the form to null. To do this you can use the following method:

```
Do $cinst.$setcurfield('')
```

When the end user taps the 'Done' button in the keyboard an `evClick` event is generated, so you can use the above code in the `$event()` method for a field to cancel the edit focus and close the keyboard.



Displaying Numbers

Single line edit fields can display number variables as well as character based data. If an integer variable is assigned to an edit field (in \$dataname), the field will use a numeric keyboard when it gets the edit focus.

If a numeric variable allows decimal points, such as Number 2dp, the standard keyboard is opened, but on the numeric page, and will only allow the input of numbers and the decimal point character (in this case, no other characters can be inserted).

iSlider Control

The Slider is a convenient and intuitive way for the end user to select a value or setting, such as a numeric value or percentage, since the position of the button corresponds to the current value of the variable assigned to the control.



The \$sliderstyle property sets the style of the slider bar. The \$min and \$max properties specify the minimum and maximum values for the slider range, while \$val is the current value in the slider range (between \$min and \$max). When the end user drags the thumb button and the slider is changed, the evSliderChanged event is triggered with pValue containing the current value.

Events

evSliderChanged	The slider value has changed; with pValue = the new value
-----------------	---

The following method can be placed behind a slider control to set the \$alpha property of an image field called oPic.

```
On evSliderChanged
    Calculate $cinst.$objs.oPic.$alpha as pValue
```

iSwitch Control

The Switch component provides an ON / OFF button. The instance variable specified in \$dataname will be set to value=1 when the end user pushes the switch on, and set to zero when the switch is turned off.



Events

evClick	a user generated click
---------	------------------------

The following method can be placed behind the Switch component to trap whether the switch is on or off and run some code as appropriate.

```

On evClick
  Switch iSwitch      ;; iSwitch (Short int)
    Case 1 ;; switch is on
      ; Do something
    Default ;; switch is off
      ; Do something else
  End Switch
    
```

iTabbar Control

The Tabbar component allows the user to select a tab which can correspond to a specific option in your application. You can specify the number of buttons on the tabbar, while for each button you can use one of the standard types or specify your own. The tabbar can be linked to a paged pane or subform so when different buttons are clicked the pane or subform can be changed accordingly. You can also allow the end user to reconfigure or reorder the buttons in the tabbar.

Property	Description
\$config	the tabbar configuration specified by the user, which allows you to store the configuration set by the user (using the value of the pConfig event parameter in the evConfigDone event)
\$currentobject	the number of the current button in the range 1 - \$objectcount; setting this in design mode allows you to set the button specific properties for each button
\$linkedobject	name of a subform or paged pane to link to the tabbar. If this is not empty, when the user selects an object, it sets \$classname or \$currentpage for the linked object to \$objectlink instead of generating a click event
\$moveobject	moves the button by assigning a number in the range 1 to \$objectcount, which changes \$currentobject to the assigned number; this provides a convenient method for you to move buttons (in design mode only)
\$objectbadge	the badge text used to mark the item; this can be useful for adding information to the button to help the user make a selection
\$objectcount	the number of buttons on the tabbar (this must be at least one)
\$objectenabled	if true, the button is enabled and generates a click event when pressed
\$objectflags	a combination of the kiTabbarButtonFlagInitiallyVisible and kiTabbarButtonFlagAlways constants that control when the tabbar button is visible in its initial state prior to the user configuring the tabbar
\$objecticonid	the icon displayed on the current button (only significant if \$objecttype is kiTabbarButtonTypeUser)

Property	Description
\$objectlink	When the user selects this object, the object sets \$classname or \$currentpage for the object to \$objectlink; only significant if \$linkedobject is not empty
\$objecttext	the text displayed on the current button (only significant if \$objecttype is kiTabbarButtonTypeUser)
\$objecttype	the type of the button specified by one of the kiTabbarButtonType.. constants; to define your own tabbar button use kiTabbarButtonTypeUser and set \$objecttext and \$objecticonid
\$openconfig	set this to kTrue to open a dialog for the user to customize or reorder the buttons in the tabbar. On completion, evConfigDone is sent with the new configuration in pConfig and invisible items in pMoreList. Changes to screen orientation are disabled while the dialog is displayed
\$selectedobject	the number (1 - \$objectcount) of the currently selected button, or zero if no button is selected

Events

evClick	a user generated click; pTabbarButton = the number of the tabbar button clicked
evConfigDone	pConfig = the new configuration of the tabbar specified by the user which is a row containing button IDs in the new order pMoreList = 2 column list; col1 is the icon ID, col2 the text

Button Icons

For most tabbar buttons you can use the standard button types available in \$objecttype and specified using one of the kiTabbarButtonType.. constants. You can however specify your own tabbar buttons using the kiTabbarButtonTypeUser object type. In this case, the \$objecttext and \$objecticonid properties are enabled which allow you to define your own button, including your own text and icon. Note that you must use icons that include Alpha properties for tabbar buttons, since the alpha values in the icon image are used to render the button image, and in this special case, the color values in the image are ignored.

If you use icons from an icon datafile or #ICONS in your tabbar, you must specify the icon page for the icon(s) in the \$iconpages property of the remote form.

Tabbar position and size

In addition to setting the tabbar specific properties, you can set the \$edgefloat property of the control to kEFposnTopToolbar or kEFposnBottomToolbar to position and “fit” the control to the top or bottom of the form, respectively.

Most of the standard tabbar buttons require a minimum height to display the icon and text correctly. In this case, you will find that you cannot resize the height of the tabbar below 49 pixels, although you can set \$height of the control if you need to specify a height.

Tabbar configuration

When you create the tabbar control in development mode, the configuration or order of the tabbar buttons is stored in the object, but at runtime you can allow the end user to change the order of the buttons. By setting the \$openconfig property to kTrue, you can open a configuration dialog on the client, allowing the end user to drag the buttons to reorder them. The following method could be placed behind a standard button to open the configuration dialog:

```
On evClick
  Do $cinst.$objs.oTabBar.$openconfig.$assign(kTrue)
```

When the end user has finished reordering the tabbar and clicked the Done button, the evConfigDone event is sent to the tabbar with the new configuration in pConfig, which is a comma-separated row containing the button numbers in the new order.

You can save the new configuration into the \$config of the tabbar using the following method placed behind the tabbar itself:

```
On evConfigDone
  Calculate $cinst.$objs.oTabBar.$config as pConfig
```

If you wish to store the new configuration permanently you will need to store the new values returned in pConfig onto the server, together with the ID of the device, and load it when the individual client device reconnects. See the *Getting App and Device Information* section earlier in this document.

Example Tabbar

Consider the Scoreboard remote form in the Anagrams example application. A tabbar is used to allow the end user to select which set of scores to display in the scoreboard.



The \$objectcount property is set to 7 to display seven tabs, with a text value for each tab specified in \$objecttext; you have to set \$currentobject to edit the properties of each tab.

The \$objecttype property controls the type or style of each button in the tabbar, and in this case each tab is set to kiTabbarButtonTypeUser which means the button is user defined. Many of the other types provide a predetermined button or icon for specific purposes.

The tabbar reports an evClick with pTabbarButton containing the number of the clicked tab which you can use to trigger an action depending on the code in the \$event() method for the control. For example, in the Anagrams example app the tab number is used to return a list of scores for the selected number of letters.

```

; contains instance vars iList (list), iPath (Char)
; iSess (Object based on OMSQLSESS), iStmt (Object no subtype)
On evClick
  Do iList.$clear()
  Do iSess.$logon(iPath, '', '', 'ScoreSess') Returns #F
  Calculate lLetters as pTabbarButton+2
  ; pTabbarButton +2 since the tabbar starts on '3 letters'
  Do iStmt.$execdirect('Select * From Scores where
Letters=@[lLetters]') Returns #F
  Do iStmt.$fetch(iList, kFetchAll)
  Do iSess.$logoff()
  Do iList.$sort($ref.RealTime, kFalse) Returns #F      ;; Sort into
time order although the time is not displayed
  Do $cinst.$redraw()

```

Note that when the data is fetched from the database, in this case an Omnis datafile, the form needs to be redrawn using the \$redraw() method, which is a method of a remote form instance, referenced using \$cinst. See below for details about the table control used on the scoreboard remote form.

iTable Control

As well as providing a method for displaying lists of data, the iTable component can be used for the main interface of your iOS app, since it can provide a hierarchical set of options, complete with graphical symbols and icons, for users to navigate their way around your application. Many iOS apps use a Table based interface in preference to the menu & button based approach found in traditional desktop and web applications.

Property	Description
\$altcolor1 & \$altcolor2	the alternating row colors of the table
\$dataname	the list variable name for the content of the table
\$grouped	set to kTrue to enable grouped mode
\$iconcolumn	the column number of the list column containing the icon id of the icon to display for each row, or zero if icons are not required
\$label1bold / \$label2bold	sets label 1 or 2 to bold
\$label1column	the column number of the list column containing the first text string to display for each row
\$label1size / \$label2size	the font size of label 1 or 2

Property	Description
\$label1textcolor / \$label2textcolor	the text color of label 1 or 2
\$label2column	the column number of the list column containing the second text string to display for each row, or zero if a second text string is not required
\$rowflagscolumn	the column number of the list column containing flags which control various options for the row, with each flag's column value a kiTableCellAccessory... constant, or zero if accessories are not required
\$rowht	the default height of a row
\$tablestyle	the style of the table, a kiTableStyle... constant

If you use icons in the table you must specify the icon page for the icon(s) in the \$iconpages property of the remote form.

The iTable component is able to display both standard “flat” lists and “grouped” lists. Standard flat lists are defined as you would any other list, containing columns and simple rows of data. In this case, you assign the flat list to the \$dataname of the iTable component and define how the data is interpreted by setting its various properties, including \$tablestyle to specify how the list determines the structure of the table (one of the kiTableStyle... constants), \$iconcolumn to specify the icon, \$label1column and \$label2column for column labels, and \$rowflagscolumn.

Events

evTableRowAccessoryClicked	pRow = a reference to the list row clicked pSection
evTableRowClicked	pRow = a reference to the list row clicked pSection

Consider the Scoreboard remote form in the Anagrams example application. A table is used to display the scores for any given number of letters, selected by the user clicking on a tabbar above the table.



The iTable control itself has no methods behind it and it does not trap events since the data is built by the end user clicking on the tabbar (see the iTabbar section). However, its visual appearance has been specified using various General and Appearance properties in the Property Manager, as follows:

\$dataname	Set to the list variable iList
\$name	Scoreboard is the simple name of the control
\$altcolor1 / \$altcolor2	Set to blue and black respectively
\$label1column / \$label2column	Set to 1 and 2 respectively which correspond to the first and second columns in the list
\$tablestyle	kiTableStyleDoubleTextHorizontal

In addition to the General and Appearance properties, you can set the properties of the text labels and row background displayed in the table under the Text tab in the Property Manager.

\$font	Courier New
\$label1bold / \$label2bold	Both set to kTrue
\$label1size/ \$label2size	Set to 14 and 12 respectively
\$label1textcolor / \$label2textcolor	Set to white and red respectively

Note you can examine the \$construct() method in the rfScoreboard remote form to see how the table (list) data is built and displayed initially.

Using Grouped Lists with iTable

Together with the simple two-column table using a flat list, you can create tables containing multiple groups of data for more powerful or complex end-user selections. To create such a table, you need to enable the \$grouped property of the table and construct a list containing the grouped structure. The different groups in your table are stored in separate lists and added to the main list specified in \$dataname and used to construct the table.



The image shows a typical grouped table; each group has a heading and any number of subheadings. Under each subheading, you can display an icon and a description. The end user can scroll the list and, in this particular example, push on a line in the table to select a new window or subform.

To create a grouped list in your code, like the one shown, you define and populate a flat list containing all the lines you want to include in a single group, add that to the main list in its first column, and add a name for the group in its second column. In the following method, both 'Group' and 'iMainList' are list variables.

```

; Instance var iMainList (List)
; Local vars: Group (List), Title, Text1, Text2, IconID (Chars) and
  Flag (Number Short Int)
Do iMainList.$define(Group,Title)
; Group 1
Do Group.$define(Text1,Text2,IconID,Flag) ;; Column order as
  defined in 'Appearance' tab
Do Group.$add('WebView','Internet
  browser',k32x32+2174,kiTableCellAccessoryDisclosureIndicator)
Do Group.$add('Pictures','Loading of pictures from your
  device',k48x48+1711,kiTableCellAccessoryDisclosureIndicator)
Do iMainList.$add(Group,'Media') ;; Add this list to our main
  list, and call the group 'Media'
; Group 2
Do Group.$define(Text1,Text2,IconID,Flag)
Do Group.$add('Entry Fields','Inputting text into entry
  fields',k32x32+2115,kiTableCellAccessoryDisclosureIndicator)

```

```
Do Group.$add('Table', 'Table to display a
    list', k32x32+1603, kiTableCellAccessoryDisclosureIndicator)
Do Group.$add('Date Time', 'Using the Datetime
    roller', k32x32+2108, kiTableCellAccessoryDisclosureIndicator)
Do iMainList.$add(Group, 'Data')      ;; Add this list to our main
    list, and call the group 'Data'
```

iMainList is assigned to \$dataname of the table, and its \$grouped property is set to kTrue.

iToolbar Control

The iToolbar provides a row of buttons to allow the end user to select an option in your application. You must specify the properties for the toolbar control itself, and for each button in turn by setting \$currentobject.

Property	Description
\$events	enable evClick to report user clicks
\$intcolor	the color of the toolbar
\$toolbarstyle	specifies the appearance of the toolbar

The button properties are as follows.

Property	Description
\$currentobject	the number (1 - \$objectcount) of the current button (this specifies the button affected by button specific properties)
\$moveobject	moves the button by assigning a number in the range 1 to \$objectcount, which changes \$currentobject to the assigned number (not assignable in class notation)
\$objectcount	the number of toolbar buttons (this must be at least one)
\$objectenabled	if true, enables the button and generates a click event when the user presses it
\$objecticonid	the icon for the current button (only significant if \$objecttype is kiToolbarButtonTypeImage), the alpha values in the source image are used to create the image, and opaque values are ignored
\$objectselected	if true, the object is the currently selected object in the group of kiToolbarButtonTypeSegmented objects to which it belongs (ignored unless \$objecttype is kiToolbarButtonTypeSegmented)
\$objectstyle	specifies the style of the button (only significant if \$objecttype is kiToolbarButtonTypeText or kiToolbarButtonTypeImage)
\$objecttext	the text on the current button (only significant if \$objecttype is kiToolbarButtonTypeText or kiToolbarButtonTypeSegmented)
\$objecttype	indicates the type of the button
\$objectwidth	the width of the button in pixels, if zero, the control automatically sizes the button (ignored unless \$objecttype is kiToolbarButtonTypeText,

Property	Description
	kiToolBarButtonTypeImage or kiToolBarButtonTypeFixedSpace)

If you use icons in your toolbar you must specify the icon page for the icon(s) in the \$iconpages property of the remote form.

Events

evClick	a user generated click; pToolBarButton = the number of the toolbar button clicked
---------	--

The following toolbar has two buttons allowing the end user to select images from their Photo Albums collection or Saved Photos folder.



The \$event() method behind the toolbar can detect which button is clicked using the pToolBarButton event parameter and act accordingly. For example, a toolbar with 3 buttons could have the following code:

On evClick

```
Switch pToolBarButton ;; contains the number of the clicked button
  Case 1
    ; Do this
  Case 2
    ; or do this
  Case 3
    ; or this
End Switch
Do $cinst.$redraw()
```

iWebView Control

The iWebView component allows you to display a web page and/or run some Javascript. You can load or reload a web page, perform a Forward or Back web command, or run some Javascript by assigning a 2 column row variable to \$execcommand to execute a command. Column 1 of the row variable must contain an integer or a kiWebViewCommand... constant, while column 2 has info specific to the command, as follows:

kiWebViewCommandLoadPage	loads the page with the URL specified by column 2 of the row assigned to \$execcommand. Sends evResult when the page has loaded
kiWebViewCommandReloadPage	reloads the current page displayed by the control. Set column 2 of the row to an empty string. Sends evResult when the page has reloaded
kiWebViewCommandForward	navigates to the next URL in the sequence of visited URLs. Set column 2 of the row to an empty string. Sends evResult when the page has loaded
kiWebViewCommandBack	navigates to the previous URL in the sequence of visited URLs. Set column 2 of the row to an empty string. Sends evResult when the page has loaded
kiWebViewCommandRunJavaScript	runs the JavaScript specified in column 2 of the row. Sends evResult when the script finishes execution

When the page has loaded, reloaded, or the Javascript has executed, the control sends the evResult event which contains the pResult parameter, a 2 column row containing the status and result of the command executed.

Events

The evResult event is sent to the iWebView control when the command assigned to \$execcommand finishes executing. The pResult parameter is a row variable, where column 1 contains the value of a kiWebViewStatus... constant (see below), and column 2 contains either error information or the URL of the page that has been loaded or the result of running the JavaScript.

kiWebViewStatusOk	indicates the command completed successfully, for kiWebViewCommandRunJavaScript, the result of the script is in col2 of pResult, for all other commands, the loaded URL is in col2 of pResult;
kiWebViewStatusLoadInProgress	means the command could not be executed because the control is currently loading content;
kiWebViewStatusLoadError	means an error occurred during command execution, column 2 of pResult contains a character string which further describes the error;
kiWebViewStatusInvalidParameter	means the parameter value (passed to the command using column 2 of the value assigned to \$execcommand) is invalid;
kiWebViewStatusCannotExecute	means the control cannot go forward or back to the next or previous URL

The following remote form contains a iWebview control called oBrowser together with a Searchbar and Segmented control to allow the user to view web pages within your Omnis application.



There is no specific code behind the oBrowser control, rather the code to load or search for pages is behind the Segmented or Searchbar controls. The following code is placed in the \$event() method for the Segmented control; the method sets a row variable depending on which segment was clicked and the contents of the row variable is assigned to the \$execcommand property of the oBrowser Webview control.

```
; lRow (Row var), Command (Short int), Value (Char)
On evClick
  Do lRow.$define(Command,Value)
  Calculate lRow.Value as ''
  Switch pSegment ;; the segment clicked
    Case 1      ;; Back Pressed
      Calculate lRow.Command as kiWebViewCommandBack
    Case 2      ;; Forward Pressed
      Calculate lRow.Command as kiWebViewCommandForward
    Case 3      ;; Refresh Pressed
      Calculate lRow.Command as kiWebViewCommandReloadPage
    Case 4      ;; Home pressed
      Calculate lRow.Command as kiWebViewCommandLoadPage
      Calculate lRow.Value as 'http://www.tigerlogic.com/omnis'
  End Switch
  Do $cinst.$objs.oBrowser.$execcommand.$assign(lRow)
  Do $cinst.$redraw()
```

Creating an Omnis iOS app

In order to test and deploy your application you need to create an iOS app which can be installed on your own iOS device or an end-user's device. This can only be done on a Mac OS X computer using the tools from Apple. To get all the necessary tools, you must sign up to the **iOS Developer Program** with Apple, at the following address:

❑ <http://developer.apple.com>

If you wish to deploy your applications onto iOS devices, you must sign up to one of the paid options under the iOS Developer Program, as you will require the code signing certificates and provisioning profiles available via the program. You can create and test your applications using the iPad or iPhone simulator, which are available if you sign up to the free version of the iOS Developer Program.

When you are ready to deploy your application you will need to consider which distribution method within the iOS Developer Program is appropriate to your app and target market.

The following section describes how you can build your Omnis iOS app for testing on the simulator or your iOS device itself.

Configuring Xcode

- Go to the Apple website and log into the iOS Developer Program
- Download and install the latest Xcode with the iOS SDK (which also provides you with the iPhone/iPad simulator)
- Go to the Omnis website (www.tigerlogic.com/omnis) and download the clientbuild.zip file
- Extract the contents of clientbuild.zip to your computer
- Locate the clientios.xcodeproj and open it in Xcode

For *all* builds, whatever the target, you should select “*Distribution*” as the “*Active Configuration*” in the “*Overview*” droplist.

Creating a Simulator App

Use the ‘omnisios_simulator’ target in the clientbuild project to build a Simulator App. You will have to setup the iOS app using the Settings section in the simulator itself as described below. The app is placed in the ../clientbuild/build/Distribution-iOSSimulator folder.

You can sign up to the iOS Developer Program free of charge to create an app that you can run on the simulator.

Creating a Device App for testing on your device

You must sign up to one of the paid options in the iOS Developer Program to create an app that you can deploy to physical iOS devices. When you have signed up to the program you can create your code signing certificate(s) and provisioning profile(s) which you will need to create a device app.

Use the ‘omnisios_device’ target in the clientbuild project to build an app for running on your iOS device (rather than the simulator). The Omnis iOS application file is called ‘Omnis.app’ and is placed in the ../clientbuild/build/Distribution-iosdevice folder. This must now be installed on the iOS device.

You will have to setup the iOS app using the Settings section on your device itself as described below.

Note that the Simulator and Device targets contain a “Settings.bundle” which will add an Omnis entry in the “Settings” on your device; this is required to allow you to setup your app for testing.

Installing your Omnis iOS app on your device

Assuming you have created the ‘Omnis.app’ file, you will need the iTunes application, available to download from Apple, to install your iOS app onto your device for testing.

- Install iTunes, if you don’t already have it, from www.apple.com/itunes
- Connect your iOS device to your computer via a cradle or USB cable, and let it sync with iTunes
- Locate your mobile provisioning profile and drop it onto your device in iTunes
- Locate your Omnis.app which you built in the previous section, and drop it onto your device in iTunes
- Then sync your iOS device with iTunes, and your Omnis app will be installed on the device

Your Omnis iOS app should now be installed on your device, and the app should appear in the list of apps on your device. However, before you can use it, you must configure the app.

Testing your Omnis iOS app

Having installed the Omnis iOS app onto your device you need to configure it to enable it to connect to Omnis Studio and your Omnis library. For developing and testing your iOS app, your device needs to connect to your local development computer running Omnis Studio and your library.

Note that for end-user deployment, your iOS app needs to connect to the Omnis Application Server which will usually be at a remote location: furthermore, clients are setup using the config.xml settings file, not using the following method.

To setup your App for development and testing

- Click on “Settings” on your iOS device, and scroll down to the Omnis entry, and select it
- Set Design Mode “Enable” to ‘On’

This allows you test your form on your local computer in the usual manner using the Test Form option or Ctrl-T.

- In the ‘IP address:Port’ field enter the IP address of the computer you are using for development and the \$serverport number of your local copy of Omnis Studio in the format IP Address:Port, e.g: 193.456.82.311:5555

Note your iOS device needs to be able to connect to a wireless network that can communicate with this IP address and port

IP address

To obtain the IP address of your development computer under Mac OS X, open the "System Preferences" dialog, click on "Network" under the "Internet & Network" section, then select your network (Ethernet, AirPort, etc) and click on the TCP/IP option.

Under Windows, you can run the ipconfig command at the command prompt.

Omnis Server Port setting

The Omnis Server port setting is in the main Omnis preferences (\$root.\$prefs). To view the Omnis preferences, click on "Studio 5.x.x" in the top left corner of the Studio Browser window, and click on "Prefs" to open the Omnis preferences in the Property Manager. Click on \$serverport to view or set the server port for your local copy of Omnis Studio. It can be any integer number in the range 1 to 32767, but we advise you to use a four digit number over 5000 to avoid any of the standard ports in use.

You can also set the Saved state delay in seconds here in the Settings file, but for initial testing you are not required to set this.

Testing your iOS remote form

To test your iOS remote form, you can press Ctrl-T in Omnis when your remote form is the top window, or you can Right-click on the background of the remote form and select "Test Form".

IMPORTANT: In order for the Test Form option to work, the Omnis iOS client *must be open and running* on your device. If the remote form and your device are setup correctly, the remote form will open on your device.

Debugging your iOS app

As with any Omnis development, you can set breakpoints in your code and step through any methods that you want to debug. So for example, you can set a breakpoint in your iOS remote form and when the breakpoint is encountered method execution is halted and the app is paused on the client. You can switch to Omnis on your development computer and step through your code. When you have executed all the methods on the stack, execution is passed back to the client.

Multi-tasking and form persistence

If you are running iOS 4.x and your device allows multi-tasking (see Apple's web site for details of which devices allow multi-tasking), your app will enter the background rather than terminate when you press the Home button. In this case, when you start the app again it will resume where it left off, because it is still running in the background; if a message was being sent to the server when you pressed the home button, the app will resend it when it comes back to the foreground.

If you are running an earlier version of iOS or if your device does not allow multi-tasking, then pressing the home button will terminate the app.

Saved State

When you close the Omnis app on your device, as part of the terminate processing, the app saves the exact state of the form, its controls and their data in a file on the device. At the same time, Omnis retains the remote form instance on the Omnis Server (or in your local

copy of Omnis during development). If you then re-open the Omnis app, the form is reinstated at the same stage as when you closed the app. This “saved state” stores the exact state of the controls, any data you or the end user has entered, and so on, so when you restart the client, the form appears exactly as when you closed the app. Remember that the remote task instance on the server has timeout properties, which could mean that the server instance has closed when the app eventually reopens.

Saved State timeout

You can set a timeout to control whether or not the current state of the remote form is saved at all or for how long the state is saved. The timeout is an integer value in seconds. If it is zero, the state of the remote form is not saved, and the application attempts to disconnect when it is closed.

The saved state for a remote form contains the time at which it was saved. If the difference between the time when the application restarts and the time in the saved state is greater than or equal to the timeout, the application being restarted discards the saved state.

To set the timeout, you can use one of the following methods.

- ❑ For design mode testing, you can set the saved state timeout for the Omnis app in the ‘Settings’ on the device.
- ❑ For a deployed application, the following member appears in the <Application> element in config.xml:

```
<iPhoneSavedStateTimeout seconds="nnn"/>
```

- ❑ In your code, you can use the new \$clientcommand method, as follows:

```
Do $cinst.$clientcommand("setsavedstatettimeout",row(timeout in
seconds))
```

The timeout in force (from the config file or set later by calling \$clientcommand) when the application saves the state is part of the saved state.

Deploying your Omnis iOS app

When you deploy your application to your target audience or market, the application needs to start automatically. In other words, you want users to be able to open your app without having to change any settings. In this case, a configuration file is included within your iOS app which contains all the necessary settings to connect the client app to the Omnis Server running your Omnis library. The “Settings.bundle” available in the ‘Simulator’ and ‘Device’ targets is not present in the ‘Distribution’ target which is used to create your release application.

For information about setting up your Omnis Server, your Omnis application (library), and Web Server for remote deployment of your Omnis iOS app, please refer to the *Extending Omnis* manual, available to download from the Omnis website and on the Omnis product DVD.

Creating a Configuration file

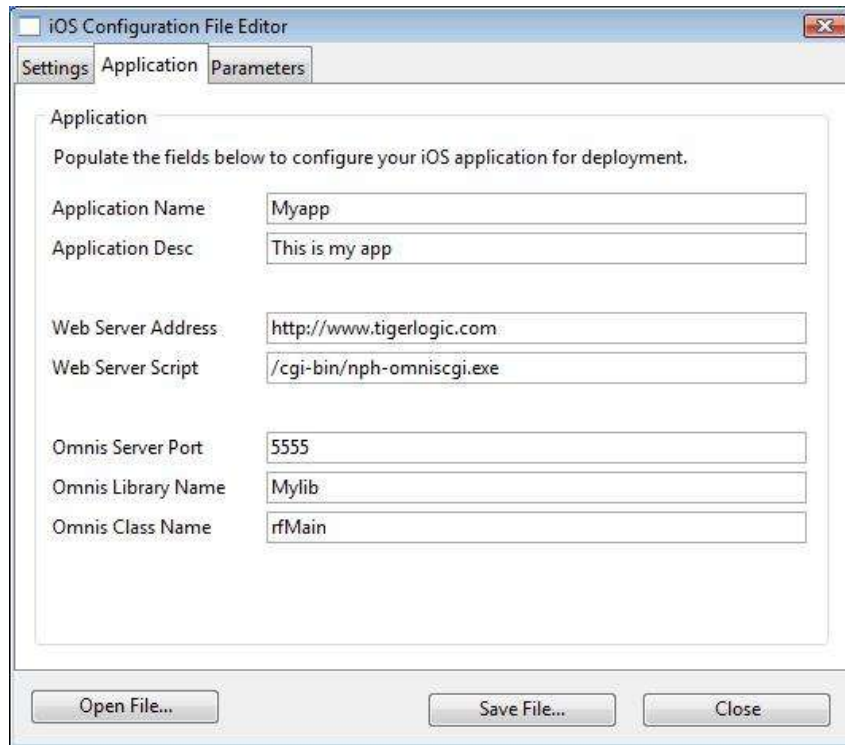
You can create a configuration file for deployment using the 'iOS Config Editor' available from the 'Add Ons>>Web Client Tools' option in the Tools menu. The dialog lets you specify the name of your Omnis library, the name of the remote form, the name and location of the Omnis Server, and so on and creates the correct structure for the file.

The following parameters are on the **Settings** tab of the editor:

- Title Bar Text**
the text shown in the title bar on the device; the default name is Omnis Studio if you leave this field blank.
- Disable Title Bar**
option allows you to disable the title bar on the device;
- Portrait only**
the default is off which means your app will respond to the device being tilted and the orientation of your remote form will switch automatically; enabling this option will only allow forms in Portrait mode
- Saved state timeout**
specifies the time in seconds that the application "saved state" will be stored, the default is 300 seconds; setting this to zero means the saved state is not stored at all
- Awaiting Connection Message**
the Message for when the application is waiting for a data connection, e.g. Waiting for data connection

The **Design Mode Server** settings let you specify a server for testing your app against a 'local' server. You should leave these settings blank for end user deployment (for a Distribution build) to ensure the app connects to your remote Omnis Server specified on the Application tab.

- Server Address** (Design/Test server only)
the IP address of the server or PC running the Omnis Server, which for testing can be a server within your local network including your own PC running the development version of Omnis Studio; note your iOS device needs to be able to connect to a wireless network that can communicate with this IP address and port
- Port**
the port number of the Omnis Server, which for testing can be your development copy of Omnis



The **Application** tab lets you specify the settings for your deployed application.

- Application Name**
The application name
- Application Desc**
The application description
- Web Server Address**
specifies the address of the Web Server through which your iOS client application connects to the Omnis Server; this can be a URL or IP-address:Port-number.
- Web Server Script**
specifies the location of the Omnis web server plug-in on your web server, such as omnisapi.dll for IIS servers; the server plug-in would typically be in the /cgi-bin, /scripts, or /webapps folder depending on your web server and operating system, e.g. /cgi-bin/omnispnph-cgi.exe.
As a special case, you can use /webclient in the script property to use the web server built into Omnis itself, which may be sufficient to run low volume apps that are accessed by a relatively small number of users at any one time.
- Omnis Server Port**
the Port number or IP-address:Port of the Omnis Server running your Omnis application (library)

- ❑ **Omnis Library Name**
The name of the Omnis library running on the Omnis Server

- ❑ **Omnis Class Name**
The name of the remote form class that clients connect to initially

The **Parameters** tab in the editor lets you add up to nine parameters to be passed, in a row variable, to the \$construct method in the remote task associated with the initial remote form in your application. Leave these fields blank if no parameters are setup in your application.

When you have completed the settings in the editor, save the file using the default name "config.xml". The configuration file is an XML file which you need to place in the root of the clientbuild folder.

The iOS Config editor allows you to load an existing config.xml to edit the file, or you can edit the file directly using an XML or standard text editor. The configuration file has the following structure, which closely corresponds to the information entered in the config editor:

```
<?xml version="1.0" encoding="UTF-8"?>
<OmnisMobileConfig>
  <WindowClass name="TIGERLOGICCORP_OMNISWM"/>
  <!-- leave WindowClass unchanged -->
  <TitleBar disable="no" text="Title bar text"/>
  <iPhoneDesignModeServer address="" port=""/>
  <!-- iPhoneDesignModeServer is for test server only-->
  <iPhoneOrientation portraitOnly="no"/>
  <WaitingForDataConnection message="Waiting for a data
connection"/>
  <Application name="Myapp" desc="This is my app">
    <ServerType unicode="yes"/>
    <!-- leave ServerType unchanged -->
    <WebServer url=" www.tigerlogic.com"
      script="/cgi-bin/nph-omniscgi.exe"/>
    <OmnisServer address="5555" library="Mylib" class="rfMain"/>
    <iPhoneSavedStateTimeout seconds="300"/>
    <Parameter name="param1" value="value1"/>
    <!-- Add up to nine parameters for the application here -->
  </Application>
</OmnisMobileConfig>
```

The **WindowClass** is the class name for application window and should remain unchanged. Only one window with this name can exist for the entire client system.

TIGERLOGICCORP_OMNISWM is reserved for use by the Omnis client. The **ServerType** specifies that the Omnis Server should be Unicode compatible, and for Omnis Studio 5.x servers should be unchanged. Note the **iPhoneDesignModeServer** property is for test servers only and should be empty for distribution builds.

Creating a Device App for Distribution

Assuming you have signed up to the iOS Developer Program and received your code signing certificate(s) and provisioning profile(s), you need to build your Omnis iOS app for distribution. For information about building apps using the certificate and provisioning profiles, please see the documentation available via the iOS Developer Program website.

Use the ‘omnisios_device_distribution’ target in the clientbuild project to build an app for deploying on end-user iOS devices. The Omnis iOS application file is called ‘Omnis.app’ and is placed in your ../clientbuild/build/Distribution-iosdevicedistribution folder.

Having created your Omnis iOS app you need to distribute it using one of the standard distribution methods provided by Apple via the iOS Developer Program.

Troubleshooting

The following are possible problems when developing your Omnis iOS app.

When I press Ctrl-T, my App does not appear on my iOS device

- a) Check that you have the Omnis app running on your iOS device. If you try to test your form without the Omnis app running, you will get the message “There is no iOS client connected to test the form”. The Omnis app *must be open on your mobile device* when you press Ctrl-T in your development copy of Omnis Studio.
- b) Check that you have installed the Omnis app on your device. Also check that you have added the provisioning file.
- c) If the Omnis app is open on your device, but your form still will not open, check the Settings on your device are correct, including the IP Address:Port setting which should point to your copy of Omnis.

The Omnis iOS app is open and running on my device, and the settings are correct, but I still cannot connect to my Omnis app.

When you want to test a form, your iOS device must have an internet connection (run Safari and check you can connect to the Internet) and your local development copy of Omnis Studio and your library *must be running*. You must select your iOS-enabled remote form in Omnis and press Ctrl-T to test your form.

When I set \$designshowmobiletitle to false (to hide the title bar) a gray bar appears at the bottom of my remote form.

You need to change the \$height property of the remote form to take into account whether or not the window title bar is visible. If you hide the title bar you need to add 20 pixels to the \$height of the form (for both Portrait and Landscape screen sizes/layouts as appropriate).

I have added an icon to a button (or some other component that accepts an icon), but when I test the app on my device the icon does not appear.

You have to add the name of the icon page containing any icons that you have used in your remote form to the \$iconpages remote form property; the icon pages listed in \$iconpages are sent to the client so if the icon pages you have used are not listed, their icons will not be displayed on the client.

My remote form is entirely covered with controls so I am unable to click on the background to set the form's properties. How can I select the form background in this case?

To select the form, you can use the Field List (right-click anywhere on the form, open the Field List and check the form name to open the Property Manager for the form), or if you click on any individual component, then shift-click it to deselect it, the focus will be returned to the form.

Amazon DAM

There is a new Omnis DAM, the Amazon DAM (DAMAZON), that allows you to access the SimpleDB from Amazon Web Services LLC. According to Amazon, "SimpleDB is a highly available, scalable, and flexible non-relational data store that offloads the work of database administration. Developers simply store and query data items via web services requests, and Amazon SimpleDB does the rest."

For further information about Amazon SimpleDB, please refer to the Amazon SimpleDB website:

❑ <http://aws.amazon.com/simplydb>

Including the Amazon SimpleDB Developers Guide:

❑ <http://docs.amazonwebservices.com/AmazonSimpleDB/latest/DeveloperGuide/>

This section also discusses various topics which differentiate cloud-based connectivity from traditional RDBMSs and the impact this has on the various properties and methods.

Dependencies

The Amazon DAM has runtime dependencies on several other dynamic libraries which must be present on your system's library search path before the DAM can be used. When a DAMAZON session object is created, the DAM attempts to locate and resolve the symbols it needs from each of the external libraries.

If one or more symbol references cannot be resolved, these are reported to the Omnis trace log as warnings, \$logon() is disabled and you should not attempt to call session or statement methods, otherwise a crash may occur.

The additional files required by the Amazon DAM for each platform are as follows:

Windows

libcurl.dll (requires msvc90.dll)

libeay32.dll (requires msvcr7.dll)

libxml2.dll (requires iconv.dll & zlib1.dll)

Mac OSX

libcurl.dylib (where libcurl.dylib -> /usr/lib/libcurl.4.dylib, for example)

libcrypto.dylib (where libcrypto.dylib -> /usr/lib/libcrypto.0.9.7.dylib, for example)

libxml2.dylib (where libxml2.dylib -> /usr/lib/libxml2.2.dylib, for example)

Linux

libcurl.so (/usr/lib/libcurl.so)

libcrypto.so (/usr/lib/libcrypto.so)

libxml2.so (/usr/lib/libxml2.so)

If these libraries are not present on your system, the appropriate package(s) may need to be installed or alternatively, downloaded and compiled from source. The principal libraries shown are all available under open source licence agreements.

For developers interested in downloading and compiling client libraries from source, information about each of the projects can be obtained from:

libcurl: <http://curl.haxx.se/>

libxml2: <http://xmlsoft.org/>

libcrypto/libeay32 : <http://www.openssl.org/> (Links accurate at time of publishing)

Binary releases of these libraries may also be available to download from these and other sources.

Logging on to SimpleDB

To connect to SimpleDB, the *endpoint* required is supplied via the \$logon() hostname parameter. In the case of Amazon SimpleDB, the endpoint is “**sdb.amazonaws.com**” or “**sdb.eu-west-1.amazonaws.com**” in Europe.

Your *access key id* and *secret* are supplied via the username and password parameters, for example:

```
Do SessObj.$logon('sdb.amazonaws.com',' AGIBJ5LOYFITD3BR7','  
H/z6t3ARzuJL26uIE07 GTS1AkK+p5') Returns #F
```

For other databases, the endpoint may be specified using http syntax, for example:

```
Do SessObj.$logon(  
  'http://www.remoteserver.com/?','user_1','password') Returns #F
```

If the hostname parameter is omitted, i.e. substituted with a comma, the DAM uses sdb.amazonaws.com by default.

Meta Data

SimpleDB does not provide information about tables, columns and indexes in the same way as traditional relational databases. Instead, *domains* can be likened to tables; *items* can be likened to rows and *attributes* can be likened to columns. This has an impact on the behaviour of the following meta-data methods:

\$Tables()	StatObj.\$Tables() returns a list of available domain names in the TableOrView column of the result set. Other result columns can be ignored as SimpleDB does not support views.
\$Columns()	StatObj.\$Columns(cDomain) returns meta data information about the specified domain. This information is specific to SimpleDB and is returned via the DamInfoRow column of the result set. Other result columns can be ignored.
\$Indexes()	StatObj.\$Indexes() is not implemented since SimpleDB handles indexing automatically.

The information returned by \$Columns() for a domain is summarised as follows:

Timestamp	The date and time when metadata was calculated in Epoch (UNIX) time.
ItemCount	The number of all items in the domain.
AttributeNameCount	The number of unique attribute names in the domain.
AttributeValueCount	The number of all attribute name/value pairs in the domain.
ItemNamesSizeBytes	The total size of all item names in the domain, in bytes.
AttributesValuesSizeBytes	The total size of all attribute values, in bytes.
AttributeNamesSizeBytes	The total size of all unique attribute names, in bytes.

SimpleDB attributes and multi-values

Unlike Relational databases, SimpleDB attributes support multiple values. For example:

<i>Domain</i>	<i>Item</i>	<i>Attribute</i>	<i>Value</i>
Suits	Gents Formal Suit	Colour	Navy
Suits	Gents Formal Suit	Colour	Black
Suits	Gents Formal Suit	Colour	Grey

In addition, SimpleDB effectively supports only a single data type: Character. All data inserted into and retrieved from SimpleDB will be character data optionally encoded as UTF-8 bytes. Once fetched into Omnis, data can be assigned to typed variables as required. Such data will be automatically converted to the appropriate data type where possible.

Each item fetched from SimpleDB can potentially have a different number of attributes and attribute names. This prevents the use of Omnis Schema classes with SimpleDB since these

require rigid column names and types. When dragging a schema class onto a SimpleDB session in the Omnis SQL Browser, all that can sensibly be achieved is to create a domain with the supplied *table name*.

SimpleDB does not support SQL in the traditional sense. You cannot use \$prepare() & \$execute() or \$execdirect() to execute INSERT, UPDATE or DELETE statements as these are not supported. Instead, these statement methods can be used only to execute SELECT statements conforming to the SimpleDB SELECT syntax.

Creating a Domain

To manually create a domain (analogous to a *table*), use the StatObj.\$createdomain() method. For example:

```
Do StatObj.$createdomain('Project810') Returns #F
```

Inserting Data

To insert items and attributes into SimpleDB, use the StatObj.\$putattrib() method.

Each call to \$putattrib() inserts a new attribute-value pair into the specified domain item. (There is no need to create the item before inserting an attribute, the item is created implicitly). Since SimpleDB supports multiple attribute values, you can assign several different values to the same attribute if required. Duplicate values are ignored. For example:

```
Do StatObj.$putattrib('Project810','Materials','Tools','13mm  
Wrench') Returns #F
```

```
Do StatObj.$putattrib('Project810','Materials','Tools','Quick  
release clamps') Returns #F
```

If many attributes are to be inserted, it may be preferable to assign the domain name to the StatObj.\$domain property and the item name to the StatObj.\$item property. These parameters can subsequently be omitted in calls to \$putattrib()- and any of the other statement methods discussed below. The above example becomes:

```
Do StatObj.$domain.$assign('Project810')
```

```
Do StatObj.$item.$assign('Materials')
```

```
Do StatObj.$putattrib(, , 'Tools', '13mm Wrench') Returns #F
```

```
Do StatObj.$putattrib(, , 'Tools', 'Quick release clamps') Returns #F
```

Deleting Data

To delete items, attributes and values from SimpleDB, use the StatObj.\$delete() method.

Deleting Values

To delete a specific attribute value, the domain, item, attribute name and value should be specified. For Example:

```
Do StatObj.$delete('Project810','Materials','Timber','50x50x2.4m  
pse') Returns #F
```

Deleting Attributes

To delete an attribute including all of its values, the domain, item and attribute name only should be specified. For example:

```
Do StatObj.$delete('Project810','Materials','Timber') Returns #F
```

Deleting Items

To delete an entire item including all its attributes and values, the domain and item name only should be specified. For example:

```
Do StatObj.$delete('Project810','Materials') Returns #F
```

Deleting a Domain

StatObj.\$delete() cannot be used to delete a domain. To do this- use StatObj.\$deletedomain(). This method should be used with caution as it will permanently delete all items, attributes and values contained in the domain before removing the domain itself. For example:

```
Do StatObj.$deletedomain('Project810') Returns #F
```

Replacing Data

Whereas \$putattrib() is used to append new attributes and values, StatObj.\$replaceattrib() is used to replace all values for a specified attribute with the supplied single value. For example:

```
Do StatObj.$replaceattrib('Suits','Gents Formal Suit','Colour','Navy only') Returns #F
```

Fetching Data

The Amazon DAM uses Amazon SELECT statements to fetch multiple items. These are issued using the statement object's \$prepare(), \$execute() and \$execdirect() methods in a similar way to traditional SQL SELECT statements. The general form of a SimpleDB SELECT statement is as follows:

```
select output_list from domain_name [where expression]
[sort_instructions][limit limit]
```

The output_list can be:

- * (all attributes)
- itemName() (the item names only)
- count(*)
- An explicit list of attributes (attribute1,..., attributeN)

For further information on the SELECT statement syntax, please refer to Amazon SimpleDB Developer Guide.

Items in the result set are returned one row-at-a-time. StatObj.\$resultspending indicates whether there is a further item each time a call to StatObj.\$fetch() is made and StatObj.\$itemcount is initially set to the number of items in the response. The destination list or row variable is automatically redefined each time \$fetch() is called. For example:

```

Do StatObj.$execdirect('select * from Suits where stocklevel > 1')
  Returns #F
Repeat
  Do StatObj.$fetch(lvRow)
  ...
Until StatObj.$resultspending = kFalse

```

Retrieving an Item

You can retrieve all attributes for a specific item using the StatObj.\$getall() method. The result set (a single row) generated by this call is returned using \$fetch(). For example:

```

Do StatObj.$getall('Suits','Gents Suits') Returns #F
Do StatObj.$fetch(lvRow)

```

Retrieving Item Names

You can retrieve the names of items contained within a domain by calling the StatObj.\$getitems() method. The result is returned as a single item containing a single attribute. The item names will be returned either as a comma-separated list or as a single column list- as dictated by the \$attribcsv property.

A SELECT where-clause may be optionally specified if required, in which case only the names of items which satisfy the expression will be returned. For example:

```

Do StatObj.$getitems(,"where Colour like 'Red%'") Returns #F
Do StatObj.$fetch(lvItems)

```

Retrieving an Attribute

You can retrieve the contents of a specific attribute using the StatObj.\$getattrib() method. The result set (a single row containing a single column) generated by this call is also returned using \$fetch(). For example:

```

Do StatObj.$getattrib('Project810','Materials','Tools') Returns #F
Do StatObj.$fetch(lvRow)

```

Handling Multiple Values

When fetching data, each row returned to Omnis represents one item from the specified domain. Item attributes containing multiple values are handled in one of two ways; either as single-column lists or as comma-separated values as dictated by the StatObj.\$attribcsv property.

When \$attribcsv is set to kTrue (the default), rows fetched from SimpleDB will be defined with Character columns. Attributes (columns) with multiple values will be returned as a string of comma-separated values.

When \$attribcsv is set to kFalse, rows fetched from SimpleDB will contain single-column lists in each column. Each single-column list will contain one row for each attribute value.

Handling Multiple Attributes

You can put, delete and replace several attribute values at once using the StatObj.\$putmany(), StatObj.\$deletemany() and StatObj.\$replacemany() methods. The attribute-value pairs to be processed are supplied via a list variable defined with two

character columns. Column 1 contains the attribute names, column 2 contains the corresponding values. For example:

```
Do myList.$define(lvChar1, lvChar2)
Do myList.$add('Tools','Posidrive screwdriver')
Do myList.$add('Tools','Metal hammer')
Do myList.$add('Charges','1½ hours labour')
Do StatObj.$putmany( , , myList) Returns #F
```

You can retrieve the values of multiple attributes using the StatObj.\$getmany() method. The attribute names to be retrieved are supplied via a *single*-column list, for example:

```
Do myList.$define(lvChar1)
Do myList.$add('Tools')
Do myList.$add('Materials')
Do myList.$add('Charges')
Do StatObj.$getmany( , , myList) Returns #F
```

Each subsequent call to \$fetch() returns a row containing separate attribute- either as a comma-separated-value or as a single column list, as dictated by \$attribcsv.

Handling Multiple Items

When executing queries, the StatObj.\$itemcount property is set to the number of items in the response- implying that each call to \$fetch() retrieves one item.

When the response contains only attribute values, \$itemcount will be set to zero.

Handling Multiple Requests

The SimpleDB DAM uses the transaction management features of the DAM interface to allow multiple requests to be executed as a combined batch of requests. To enable multiple-execution, the SessObj.\$transactionmode property should set to kSessionTranManual.

In this mode, actions such as \$createdomain(), \$putattrib(), \$getattrib(), \$replacemany() and \$execdirect() are accepted unconditionally into a queue. Nothing is sent to or received from the database until a SessObj.\$commit() is executed, at which point each request is submitted in turn.

Unlike single request execution, every multiple request generates a response. Although actions to put, create, replace and delete attributes will return empty responses, this enables any errors and execution information associated with each action to be returned. For example:

```
Do cSess.$begin()
Do cStat.$putattrib(,,'Materials','White Paint') Returns #F
Do cStat.$putattrib(,,'Materials','Cement 25Kg') Returns #F
Do cStat.$replacemany(, ,lvAttribList) Returns #F
Do cStat.$execdirect('select * from Project810') Returns #F
Do cSess.$commit() Returns #F
```

Handling Multiple Responses

When in manual transaction mode, each call to `$commit()` generates one or more responses. The number of responses available is returned via the `SessObj.$responses` property.

When `$commit()` is executed, `StatObj.$itemcount` and `StatObj.$columncount` are set to reflect the number of items and attributes in the initial response.

Items/attributes from the response are then retrieved using one or more calls to `$fetch()`. When all items/attributes from the current response have been retrieved, the `StatObj.$endofresponse` property is set to `kTrue` at which point, `$itemcount` and `$columncount` are also set to reflect the next response.

When fetching an empty response, note that `$endofresponse` will effectively remain set to `kTrue`. If the corresponding action generated an error, then `StatObj.$nativeerrorcode` and `StatObj.$nativeerrortext` will be set accordingly. Otherwise, the empty response (and empty row) can be discarded.

When all responses have been retrieved, the `$resultspending` property is set to `kFalse`, otherwise `$resultspending` remains set the `kTrue` while there are still responses waiting.

It is safe to abandon and/or replace multiple requests before executing them by simply calling `SessObj.$begin()` or changing the transaction mode back to `kSessionTranAutomatic`. You can also discard pending responses in this way.

`$rollback()` is not supported by the SimpleDB DAM- this has no effect.

Machine Utilization

Amazon SimpleDB measures usage of remote resources (and hence the charge it imposes on the end-user) in terms of “box usage”. Each action sent to the database incurs a box usage- quoted as a decimal fraction of one hour. `StatObj.$boxusage` returns the box usage for each action which generates a response.

The *session* object also has a `$boxusage` property which accumulates a total box usage for the open connection. When retrieving multiple responses, the box usage for each response is received (and added) in turn.

`$boxusage` may not be supported by all Simple databases in which case, the value remains set to zero.

Read Consistency

Amazon SimpleDB supports two types of read consistency, defined as follows:

- ❑ **Eventually Consistent Reads**
the eventual consistency option maximizes your read performance (in terms of low latency and high throughput). However, an eventually consistent read (using `Select` or `GetAttributes`) might not reflect the results of a recently completed write (using `PutAttributes`, `BatchPutAttributes`, `DeleteAttributes`). Consistency across all copies of data is usually reached within a second; repeating a read after a short time should return the updated data.

❑ **Consistent Reads**

in addition to eventual consistency, Amazon SimpleDB also gives you the flexibility and control to request a consistent read if your application, or an element of your application, requires it. A consistent read (using `Select` or `GetAttributes` with `ConsistentRead=true`) returns a result that reflects all writes that received a successful response prior to the read.

The Amazon DAM implements this functionality using the `$consistentread` session property. When set to `kFalse` (the default setting), the eventual consistency option is used. When set to `kTrue`, all `$getattr()` and `SELECT` statement results are fetched using consistent reads.

Conditional Puts and Deletes

The `PutAttributes` and `DeleteAttributes` API calls used by Amazon SimpleDB support conditional put and delete operations which enable you to insert, replace or delete values for one or more attributes of an item if the existing value of an attribute matches the value you specify. If the value does not match or is not present, the update is rejected. Conditional Puts/Deletes are useful for preventing lost updates when different sources write concurrently to the same item.

The Amazon DAM implements this functionality using the `$whereclause` statement property. This property affects all *put*, *replace* and *delete* attribute calls and accepts a SQL-style where clause of the form:

“where <name> [= <value>] [exists|does not exist]”

<name> and <value> can be literal values; in which case they must be double-quoted, or bind variables. Double quotes inside literal values should be escaped using `\`. For example:

```
Do cStat.$whereclause.$assign(\where "Color" = "Light Brown")
```

```
Do cStat.$whereclause.$assign(\where "Undo" does not exist')
```

```
Do cStat.$whereclause.$assign(\where "Project \"X\" = @[lvChar]')
```

Once bound, variable values should be assigned before each call to `$putattrib()`, `$delete()`, etc:

```
Do cStat.$whereclause.$assign(\where "Name" = @[lvChar]')
```

```
Calculate lvChar as "Brookes"
```

```
Do cStat.$putattrib('StockDB', 'Supplier1', 'Frequency', 'Daily')
```

```
Returns #F
```

```
Calculate lvChar as "Robinson"
```

```
Do cStat.$delete('StockDB', 'Supplier2', 'Frequency') Returns #F
```

Currently, the *exists* condition may only be specified if both <name> and <value> attributes are also specified. To use *does not exist*, only the <name> attribute should be specified.

Subsequent calls to *put*, *replace* or *delete* attributes return `kFalse` if the condition is not met.

`$whereclause` is not affected by `$clear()`. To remove the where condition for a statement object; assign `$whereclause` to an empty string.

Session Properties

Property	Meaning
\$boxusage	Returns the cumulative total of remote machine resources consumed since the session connected. Collects box-usages from statement methods as well as box-usages from multiple actions (manual transactions). Read-only.
\$consistentread	If set to kTrue, all read operations (e.g. \$getattr() & \$fetch()) are executed guaranteeing that the results of recent updates are seen immediately. If set to kFalse, the default (faster) eventual consistency option is used.
\$responses	Returns the number of responses generated by the last call to \$commit(). Applies to manual transaction mode only. Read-only.
\$transactionmode	Used to implement multiple request processing. When set to kSessionTranAutomatic each request is sent to the database immediately. When set to kSessionTranManual, requests are queued until a \$commit() is called.

Session Methods

Method	Description
\$begin()	Initialises/clears multiple responses in preparation for execution of a new batch of requests. Manual transaction mode only.
\$commit()	Executes a batch of statements and retrieves multiple responses from the database. Manual transaction mode only.

Statement Properties

Property	Meaning
\$attribcsv	If set to kTrue (the default), attributes with multiple values are returned as comma-separated values, i.e. the fetched row will be defined with character columns. If set to kFalse, attributes will be returned as single column lists, i.e. the fetched row will contain a single column list in each column.
\$boxusage	For statement methods which generate a response from the database, \$boxusage returns the portion of a machine hour used to complete a particular request. See SessObj.\$boxusage. Read-only.
\$domain	The current domain name. \$domain will be used with various statement methods if set. Statement methods which require a domain parameter will assume this value if the method parameter is omitted.
\$endofresponse	Returns kTrue if the last item/attribute of the current response has been fetched in which case, \$boxcount, \$itemcount and \$columncount are set to reflect the next response. Read-only.
\$item	The current item name. As with \$domain, \$item will be used with various statement methods if set. Statement methods which require an item name will assume this value if the method parameter is omitted. When retrieving an item list from the database, \$item is also set to the name of the last item to be fetched.
\$itemcount	Returns the number of items in the current response. Returns zero if the response contains only attribute information. Read-only.
\$resultspending	Returns kTrue while there are still items/attributes waiting to be fetched from one or more responses. Read-only.
\$whereclause	Affects all put, replace and delete attribute methods. This property accepts a SQL-style where clause of the form: “where <name> [= <value>] [exists does not exist]” <name> and <value> can be literal values; in which case they must be double-quoted, or bind variables. Double quotes inside literal values should be escaped using \”

Statement Methods

Method	Description
\$createdomain()	StatObj.\$createdomain([cDomainName]) creates a domain with the specified name. \$createdomain() uses the value of StatObj.\$domain if the parameter is omitted in which case, \$domain must be predefined. Returns kTrue on success, kFalse otherwise.
\$delete()	StatObj.\$delete([cDomain],[cItem],[cAttrib],[cValue]) deletes an item, attribute or value from the specified domain. If cDomain or cItem are omitted, the values of StatObj.\$domain and StatObj.\$item are assumed in which case, \$domain and \$item must be predefined. If cAttrib and cValue are omitted, the entire item is deleted. If cValue is omitted, the specified attribute is deleted. Otherwise, the specified value only is deleted from the attribute. Returns kTrue on success, kFalse otherwise.
\$deletedomain()	StatObj.\$deletedomain([cDomain]) deletes the specified domain and all associated items/attributes. Warning: no further confirmation is sought before the domain is permanently deleted. Returns kTrue on success, kFalse otherwise.
\$deletemany()	StatObj.\$deletemany([cDomain],[cItem],lAttribs) deletes one or more values from the domain item. The attribute-value pairs are supplied via lAttribs, which should be defined with two character columns. Column 1 contains the attribute name, Column 2 contains the corresponding value to be removed. Returns kTrue on success, kFalse otherwise.
\$getall()	StatObj.\$getall([cDomain],[cItem]) executes a query to return all attributes belonging to the specified item. The result of the query is retrieved by calling StatObj.\$fetch(). Returns kTrue on success, kFalse otherwise.
\$getattrib()	StatObj.\$getattrib([cDomain],[cItem],cAttrib) executes a query to retrieve the value(s) associated with the specified attribute. The result of the query is retrieved by calling StatObj.\$fetch(). Returns kTrue on success, kFalse otherwise.
\$getitems()	StatObj.\$getitems([cDomain],[cWhere]) executes a query to retrieve the item names contained within the specified domain. If cWhere is specified, the text is appended to the SELECT statement. The result of the query is obtained by calling StatObj.\$fetch(). Returns kTrue on success, kFalse otherwise. \$getitems() is not supported by all database vendors.
\$getmany()	StatObj.\$getmany([cDomain],[cItem],lAttribs) executes a query to retrieve one or more named attributes from the domain item. The attribute names are supplied via lAttribs, which should be defined with

Method	Description
	a single character column. The result of the query is retrieved by calling StatObj.\$fetch(). Returns kTrue on success, kFalse otherwise.
\$putattrib()	StatObj.\$putattrib([cDomain],[cItem],cAttrib,cValue) inserts a new attribute. If cAttrib already exists, the new value is appended to the existing value(s), otherwise a new attribute-value pair is created. Returns kTrue on success, kFalse otherwise.
\$putmany()	StatObj.\$putmany([cDomain],[cItem],lAttribs) inserts one or more values into the domain item. The attribute-value pairs are supplied via lAttribs, which should be defined with two character columns. Column 1 contains the attribute name, Column 2 contains the corresponding value. Returns kTrue on success, kFalse otherwise.
\$replaceattrib()	StatObj.\$replaceattrib([cDomain],[cItem],cAttrib,cValue) replaces all values for the specified attribute with the specified value. Existing values are deleted. Returns kTrue on success, kFalse otherwise.
\$replacemany()	StatObj.\$replacemany([cDomain],[cItem],lAttribs) replaces one or more attributes in the domain item. The attribute-value pairs are supplied via lAttribs, which should be defined with two character columns. Column 1 contains the attribute name, Column 2 contains the new value. Existing values are deleted. Returns kTrue on success, kFalse otherwise.

Implementation Notes

Bind Variables

Queries issued using \$execute() and \$execdirect() may contain bind variables- for example in the where clause of the SELECT statement. The DAM inlines variable values into the SQL text each time \$execute() is called, placing single quotes around each value. Values containing single quotes are escaped by adding a second single quote for each occurrence.

For example:

```
Calculate lVar as "Katharine O'Hara"
Do StatObj.$execdirect("select * from Customers where Name =
    @[lVar]") Returns #F
    becomes
select * from Customers where Name = 'Katharine O''Hara'
```

Multiple Statement Objects

The SimpleDB API does not facilitate statement isolation- only session isolation. This means that each session object may spawn only one statement object.

An attempt to spawn a second statement object will result in an error.

Remote Procedures

SimpleDB does not support remote procedures, views or triggers. These are features of traditional relational databases.

Binary Data

SimpleDB attributes support only character data of maximum length 1024 bytes and are not suitable for storing binary data directly. A better approach (the intended approach) is that attribute values be used to store URLs or unique identifiers for pictures, files and other media which exist externally to the database.

Miscellaneous Enhancements

HTTP commands

The Map+ parameter has been added to the HTTPGet() and HTTPPost() commands. When passed as kTrue (the default = kFalse), plus characters in CGI parameter names and values in the CGI List are URL encoded using hex. (ST/FU/588)

The complete syntax for the commands is now as follows:

```
HTTPGet (host,uri[,cgilist,hdrlist,service|port,secure {Default
        kFalse},verify {Default kTrue},map+ {Default kFalse}}) Returns
        socket
```

and

```
HTTPPost (host,uri[,cgilist,hdrlist,service|port,secure {Default
        kFalse},verify {Default kTrue},map+ {Default kFalse}}) Returns
        socket
```

Apple Menu Hide Key String

The Apple + H key combination is now used in Mac OS X to hide the current top window which is causing issues for Omnis applications that use this key combination for other purposes; OS X is intercepting Apple + H and the key event is never passed to Omnis. (ST/HI/1534)

The AppleMenuHideKey command has been added to the built-in strings in Omnis, so you can edit this string to change the letter to activate the command. The new string is in the format:

```
AppleMenuHideKey:H
```

You need to edit the string and change its last character. Setting the character to underscore removes the shortcut. This shortcut is applied to both the 'hide omnis' command and 'hide others' in conjunction with the option key.

Remote Menu Lines

You can now set the text for a remote menu line to \$st.id. The lookup occurs when the menu is built on the client, before any event processing for the menu. (ST/WT/1623)

What's New in Omnis Studio 5.0.1

Omnis Studio 5.0.1 provides support for Windows 7 and Mac OS X Snow Leopard (version 10.6), together with a number of other enhancements, as follows.

- ❑ **Windows 7 and Mac OS X**
Studio 5.0.1 is fully compatible with Windows 7 and Mac OS X Snow Leopard
- ❑ **Client commands**
a new method called \$clientcommand() allows you to execute various functions within the Web Client including Yes/No messages
- ❑ **Web content tips**
content tips are now supported for single line and multi-line fields in remote form
- ❑ **Vertically centered text**
new properties for window fields that allow you to center text vertically in a field
- ❑ **Tree lists**
you can now use 32x32 or 48x48 icons for tree list nodes
- ❑ **Managing Studio messages**
new sys() functions allow you greater control of certain error messages
- ❑ **Closing the Trace log**
a new command gives you the ability to close the Trace log in the Omnis Runtime or Server
- ❑ **Security for Web commands**
SSL has been added to the TCP commands
- ❑ **Email authentication**
CRAM-MD5 authentication support has been added to SMTPSend
- ❑ **Server Load Sharing**
You can now run the Load Sharing Process executable (omnislsp.exe) as a service (Windows only)
- ❑ **Omnis DataBridge**
There have been a number of changes to how the Omnis DataBridge (ODB) works under Windows

Windows 7

Omnis Studio 5.0.1 fully supports Windows 7. A few enhancements have been made to support the new platform, as follows.

Functions

There is a new function, `iswindows7()` to test for the new platform, plus there is a new parameter for the `isvista()` function to test for a theme (these functions apply to both Desktop and Web clients).

iswindows7()

`iswindows7([themed=kfalse])`

For `themed = kFalse`, returns true if the current operating system is Windows 7 or later; for `themed = kTrue`, returns true if the current operating system is Windows 7 or later with a themed appearance.

isvista()

`isvista([themed=kfalse])`

For `themed = kFalse`, returns true if the current operating system is Windows Vista or later; for `themed = kTrue`, returns true if the current operating system is Windows Vista or later with a themed appearance.

Tree lists and Icon arrays

Ctrl-click now deselects the clicked entry for multiple select tree lists and icon arrays.

Toolbars

The default toolbar background and text color have been changed to match the operating system.

Mac OS X

Omnis Studio 5.0.1 fully supports Mac OS X Snow Leopard.

Functions

There is a new function to test for Snow Leopard (this applies to both Desktop and Web clients).

issnowleopard()

`issnowleopard()` – no parameters

Returns true if the current operating system is Mac OS X Snow Leopard (version 10.6) or later.

Client Commands

There is a new remote form method called `$clientcommand()` that allows you to execute various functions within the Omnis Web Client, including several types of message boxes such as an OK message and Yes/No message. This is useful for methods that are not set to 'Execute on the Client' but may require some limited functionality to operate on the client. The new method must be executed on the Omnis Server in a remote form instance, and requires various parameters dependent on the command sent to the client. The `$clientcommand()` method has the following general syntax:

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

where `$cinst` is the current remote form instance, *commandname* is the name of the command to be executed on the client, and the *row variable* contains any number of parameters passed to the client. In the case of a message box, the row variable contains the message text, the title text for the box, and the name of methods to be called when either the Yes or No button is pressed (the method names can be null or empty if no method is to be called).

A number of commands have been implemented, but further commands may be added in the future.

Client Message boxes

The following message box functions can be used without the need for client method execution.

yesnomessage

Opens a Yes/No message box in which Yes is the default button.

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

Where *row-variable* is `row(message text, title text, name of public form server method called on Yes, name of public form server method called on No, name of public form server method called on Cancel (leave empty for no cancel button))`.

noyesmessage

Opens a No/Yes message box in which No is the default button.

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

Where *row-variable* is `row(message text, title text, name of public form server method called on Yes, name of public form server method called on No, name of public form server method called on Cancel (leave empty for no cancel button))`.

okcancelmessage

Opens an OK/Cancel message box in which OK is the default button.

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

Where *row-variable* is `row(message text, title text, name of public form server method called on OK, name of public form server method called on Cancel)`.

System Messages

soundbell

Plays the default sound on the client.

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

In this case, row() is empty.

Entry Fields

Content tips

The \$contenttip property has been added to single line and multi-line entry fields in remote forms. Content tips allow you to add text to fields to help the user understand what content should be entered.

Remote Forms

Background gradient patterns

You can now assign gradient patterns to the background of remote forms.

Window Fields

Vertically Centered Text

There is a new property, called \$vertcentertext that controls whether or not text is centered vertically in the field area. The new property makes it easier to produce well aligned text and fields across all the platforms supported in Omnis. Using this new property in your windows and remote forms will allow you to line up the base-line of text labels and the text data contained in fields.

❑ \$vertcentertext

If true, single line text is vertically centered in the height of the field. If false, the text is vertically positioned according to the default positioning for the field. For existing fields the property is set to kFalse.

The new \$vertcentertext property is available for several window field types, including single line edit fields, combo boxes, droplists, background labels, background text objects, string labels, shape fields (the text part), checkboxes (no border), radio buttons (no border), masked entry fields. The new property also applies to several remote form fields, including single line edit fields, combo boxes, droplists, background labels, string labels, checkboxes (no border), and masked entry fields. Note the property is not available for multi-line fields on windows and remote forms.

When `$vertcentertext` is `kTrue` for a label and a field, if you give the label the same height and `$stop` property as the field, the text will draw on the same base-line on all platforms, provided the field and label have the same font, point size and style.

There is a new entry on the Align menu, to align labels vertically. To align a number of labels and fields, select the objects and select the Center Text Vertically option. The new align option attempts to find labels for fields (objects that have the `$vertcentertext` property within the selection), and sets `$vertcentertext` to `kTrue`. The `$stop` and `$height` of the label is also set to match the corresponding field. You can use Undo to reverse the alignment.

If you have used a multi-line entry field with a label, the above will not work satisfactorily. To achieve the correct base-line drawing, use a label with `$vertcentertext` set to `kFalse`, and set its `$stop` coordinate to the `$stop` coordinate of the multi-line field plus the height of the top border of the multi-line field.

Tree lists

You can now use 32x32 or 48x48 size icons for tree list nodes. You need to turn on the new `$useiconsize` appearance property of the tree object to assign larger icon sizes to tree lists. You may also need to adjust the `$treelinehtextra` and `$treeindentlevel` properties to accommodate the larger icons.

Combo boxes

The `$contenttip` property has been added to combo boxes allowing you to add text to the field to help the user understand what content should be entered.

Modify Report Fields

The `$disablessystemfocus` property has been added to Modify Report Fields, so you can disable the system focus indicator in the field.

Scroll box fields

You can now apply gradient patterns to scroll box fields.

You can use this new effect in several different ways. For example, if you want to make entry fields within the scroll box appear to be transparent, you can apply a gradient to the scroll box, place the fields within the scroll box and set their background theme to parent.

Localization

String Labels

The string label object (part of the String Table package, used to enable multi-language text labels) for both remote forms and windows can now display multiple lines of text.

Tooltips

The `$t.` lookup notation now works for tooltips, that is, to lookup text from a string table to fill out the `$tooltip` property for a field.

Studio Messages

"You have been disconnected" type errors are no longer logged by default. There are two new `sys()` functions to allow you to turn these messages on and off. (ST/PF/665)

- `sys(3904)`
turns on logging for "You have been disconnected" type error messages.
- `sys(3905)`
turns off logging of these messages.

Trace Log

A new command, Close trace log, has been added to allow you to close the trace log via a method, such as within the Runtime or Server version of Omnis Studio. The new command can be executed in the Web Client.

Web Commands

TCPConnect

TCPConnect can now establish a secure connection. It has new optional secure and verify parameters. The syntax for the command is now:

```
TCPConnect (hostname,service|port[,secure {Default kFalse},verify  
{Default kTrue}]) Returns socket
```

If the connection is secure, a send using *TCPSend* will always be blocking, even if the socket is marked as non-blocking. The *TCPBlock* command makes a socket blocking or non-blocking.

See the Omnis Help (F1) under *TCPConnect*, *TCPSend*, and *TCPBlock* for further details.

SMTPSend

In Omnis Studio 5.0, the *SMTPSend* command only supported the LOGIN and PLAIN methods of SMTP authentication. The *SMTPSend* command now supports CRAM-MD5 authentication, as well as the LOGIN and PLAIN methods.

If the mail server supports CRAM-MD5, then *SMTPSend* will use CRAM-MD5 if more than one of these three methods is available, as this is the most secure method of the three it supports.

See the Omnis Help (F1) under *SMTPSend* for further details.

Server Load Sharing

Installing omnislsp as a Service (Windows only)

The Server Load Sharing Process executable (*omnislsp.exe*) can now be installed as a service which starts-up automatically when Windows loads. For this purpose, two additional parameters are supported:

<code>omnislsp -install</code>	Creates and starts the “Omnis Load Sharing Process” service.
<code>omnislsp -uninstall</code>	Stops and removes the service.

The startup-type for the new service is set to “Automatic” and the service uses the *omnislsp* executable and *.ini* file at their current locations. When *omnislsp* runs as a service, dialog boxes are disabled and messages are written to the application event log instead.

Using the ODB

There have been a number of changes to how the Omnis DataBridge (ODB) works under Windows. Specifically, the ODB is now a command line app on Windows, and the separate ODBService.exe has been removed.

The ODB manual (odb.pdf) has been updated accordingly, but the changes have also been included here for the benefit of existing users.

On Windows 2000/XP/Vista/7

Launching ODB

Once you have updated the configuration file with the details of your data files, you can launch the ODB by running the odbridge.exe located in the ODB folder.

Open the command prompt and navigate to the ODB folder.

Type

```
odbridge      (or  odbridge start)
```

And press return.

If ODB has started you should see

```
Executing start...  
...see messages.txt for success
```

You can check the file *messages.txt* in the ODB folder to check if ODB has started successfully. If an error has occurred this file will contain details of the error, or

```
ODB is now listening for requests from your clients
```

Note: You can also start the ODB by double-clicking odbridge.exe in the window view. In this case the data bridge process starts silently, although messages are still written to the messages.txt file.

Shutting down ODB

To stop the data bridge process, type

```
odbridge shutdown
```

And press return.

If ODB has received the shut down request, you should see

```
Executing shutdown...  
...see messages.txt for success
```

If there are users connected, shutdown will fail. It is not safe to shutdown ODB while users are still connected. You should ask the users to disconnect (that is, close their data files) before trying again.

However, it is possible to force the ODB to shutdown with users still connected.

Type

```
odbridge kill
```

and press return.

Note: You risk potential data file corruption if a user is writing to the data file while shutting down.

Installing as a Service

You can install the ODB as a Windows service which starts automatically each time Windows loads. To do this, make sure that the ODB is not running, then type

```
odbridge install
```

and press return.

If the installation is successful, you should see

```
Installing "C:\...\odbridge.exe" as a service
  The operation completed successfully.
```

If an error occurs, the error message will be displayed instead. By default, the service (named "Omnis Data Bridge") is configured to start automatically. You can modify this setting via the Windows Control Panel->Administrative Tools->Services panel if required.

Removing the Service

To stop and remove the "Omnis Data Bridge" service, type

```
odbridge remove
```

And press return.

If removal is successful, you should see

```
Uninstalling service: odbridge
  The operation completed successfully.
```

If an error occurs, the error message will be displayed instead.

Note: Removing the service is equivalent to executing the "odbridge kill" command. You should therefore ensure that there are no clients connected before removing the service.

What's New in Omnis Studio 5.0

Note the following information was available in the 'What's New in Omnis Studio 5.0' manual, but is reproduced here for your convenience.

Omnis Studio 5.0 contains many new features and other enhancements to further increase the power and flexibility of the Omnis development environment. The main theme for Omnis Studio 5 is 'Extending Omnis to New Platforms and New Markets':

- ❑ **New Platforms:**
support for Windows Mobile® based phones or 'Smartphones' and other mobile devices
- ❑ **New Markets:**
full support for Unicode and extended character sets means you can localize your application for virtually any market place in the world

New Features

The key new features of Omnis Studio 5.0 are listed below. There are many other enhancements described later in this document.

- ❑ **Omnis Mobile Client**
The new Mobile Client will allow your Omnis applications to run on Windows Mobile® based devices including Smartphones, PDAs, and other mobile devices. The new Omnis Mobile Client will allow you to extend your applications to a whole range of mobile devices, and create applications for entirely new markets.
- ❑ **Unicode**
In addition to the new features, all versions of Omnis Studio 5 will support Unicode, which means you can store and display data that uses Unicode based characters. Note there are no longer Unicode serial numbers – Omnis Studio 5 serial numbers make no distinction between Unicode or Non-Unicode
- ❑ **Localization**
The String Label object is now available for Remote forms which means you can localize the text and strings in your Web Client based applications. In addition, two new string tables have been introduced to allow you to translate more built-in resources (strings, messages, toolbars etc) in Omnis and the Omnis Web Client. In addition, the String Table Editor has been enhanced and now allows you to translate a whole list of strings into one or more languages automatically.
- ❑ **Menus for Web and Mobile applications**
A new class, a Remote menu class, has been added to provide context menus for

remote forms. The new remote menu can be used in web browsers or on Windows Mobile based devices.

- ❑ **Enhanced Component Interface and New Components**

The Omnis External Component interface now supports animation and transparency allowing developers to create new interactive and visually rich components. Plus we have created two new components, the Accordion and Fisheye, available for window classes and remote forms, to further enhance your application development.
- ❑ **Configurable Web Server plug-in**

Now you can configure the Web Server plug-in via a separate configuration file, allowing greater security and control over user access to your Omnis web applications.
- ❑ **VCS**

Various enhancements in the Omnis VCS including the ability to create branches in a project, the Find Class option, the ability to strip Comments from builds, the Checked out classes warning, Project Property updates, and more.
- ❑ **Method Performance**

You can now collect performance data for the methods within individual classes in your application, and method lines can now have line numbers making it easier to read your code
- ❑ **Windows Registry Admin**

The new Omnis RegAdmin component is a non-visual object that gives you access to the Windows Registry allowing you to manage Keys and Values within the registry.
- ❑ **IMAP and Security added to Web Commands**

Support for secure connections has been added to the existing package of web commands. Therefore, the HTTP, FTP, SMTP and POP3 client commands that establish a connection to a server all have two new arguments, which control if and how a secure connection is used. In addition, there is a new set of commands to allow communications with an IMAP server.
- ❑ **OpenBase DAM**

This release includes a new Omnis DAM to allow you to connect to the OpenBase database. This DAM was previously available from OpenBase, but we have taken over the maintenance of this DAM with the release of a new version of the DAM compatible with Omnis Studio 5.

Serial Numbers

The Development, Runtime and Server versions of Omnis Studio 5.0 require a new serial number to run. Serial numbers from previous versions of Omnis Studio will not work with Omnis Studio 5.0.

In addition, Omnis Studio version 5.0 serial numbers make no distinction between Unicode and non-Unicode since the Studio 5.0 release is Unicode only.

Library and Data File Conversion

Omnis Studio 5 will attempt to convert libraries and Omnis data files when you open or access them in this new version.

THE CONVERSION PROCESS IS IRREVERSIBLE, SO PLEASE MAKE SURE YOU HAVE A SECURE BACKUP OF ALL YOUR OMNIS LIBRARIES AND DATA FILES BEFORE CONVERTING THEM TO OMNIS STUDIO VERSION 5.0.

In addition to the normal conversion process, which occurs for all new major versions, Omnis Studio 5.0 will convert all libraries and data files to a **Unicode compatible format**, which again is irreversible. See the Unicode section in this document for specific information about converting your Omnis data files to Unicode.

System Requirements

This section contains information that you may need to run Omnis Studio 5.0. The basic System Requirements for running Omnis Studio 5 are as follows:

Windows

Intel system, 1GB RAM, DVD-ROM drive, and 150 MB free hard disk space, Windows Vista (2GB RAM), Windows XP SP2, Windows 2003 SP1 are preferred.

Omnis Studio is likely to run without service packs and with earlier NT-based operating systems, such as Windows 2000, however these are not tested configurations.

Linux

Intel system, 1GB RAM, DVD-ROM drive, and 150 MB free hard disk space. 128 MB Video card capable of supporting 24 bit color and a screen resolution of 1024x768. Redhat 9, SuSE 10 or Ubuntu 6.

Omnis Studio is likely to run on any Linux distributions that have version 2.0 or higher kernel, and xfree86 3.3.4 or higher.

Mac OS X

Intel system, 1GB RAM, Mac OS X version 10.4 or above, DVD-ROM drive and 250MB free hard disk space.

Omnis Studio is likely to run without OS updates, however these are not tested configurations.

Windows Mobile Client

Introduction

Computing is becoming more mobile with laptops and other handheld devices now outselling traditional desktop computers. The leading growth area in the handheld market is for so-called 'Smartphones', which have all the familiar features of mobile phones together with extended features like WiFi connectivity, satellite positioning and support for professional office applications like email, word processing, and spreadsheets. These mobile devices are becoming the must-have device for most business people, allowing them to communicate while out of the office, to have access to critical data, and to organize their work and family life more flexibly.

With the release of Omnis Studio 5, you can take advantage of this mobile revolution, by extending your new and existing Omnis applications to support these new mobile platforms and devices. Combined with all the other features in Studio 5, such as support for multiple languages (Studio 5 fully supports Unicode text and data), you can now reach new customers, in new markets, giving you greater success for your Omnis applications.

There is a short tutorial at the end of this section to show you how quickly and easily you can create or adapt remote forms for use on mobile devices.

The Omnis Mobile Client

Omnis Studio 5 will include a new version of the Omnis Web Client plug-in that supports applications running on Microsoft® Windows Mobile® based devices. The new mobile enabled client is called the **Omnis Mobile Client** and works in a very similar way as previous versions of Omnis in that it displays a *remote form* on the mobile device. The major difference with the new mobile-enabled client is that it does not run inside a web browser, such as the mobile version of Internet Explorer, rather it runs inside a small, stand-alone program, which is downloaded to the mobile device along with the client.

The remote forms that you can run in the new Omnis Mobile Client have a specific size and format and therefore need to be designed especially for mobile phones, but this is all taken care of in Omnis in design mode when you create the form. Beyond the size, designing a remote form for the mobile client is virtually the same as designing a remote form in the current version of Omnis, and you can use most of the components available in remote forms today. Your computer and mobile device are connected using Windows Mobile Device Center (previously called ActiveSync), and once set up, you can test your forms on the device itself by pressing Ctrl-T.

The Omnis Mobile Client runs on version 5.0, 6.0 and 6.1 of Windows Mobile from Microsoft. Specifically, the new client has been tested on the new breed of smartphones which run Windows Mobile 'Professional' edition. These phones have all the usual phone

capabilities, touch sensitive screens, and PDA functions including the mobile version of Word, Excel and PowerPoint.

Windows Mobile is available in various editions including Standard, Professional, and Classic. The Standard version is for mobile phones with only basic phone functionality and without touchscreens, while Professional is for Pocket PCs or PDAs with touchscreens and full phone functionality. The Classic version is for Pocket PCs without phone capabilities. The Omnis Mobile Client does not run on the Standard version of Windows Mobile.

Creating Mobile Forms

Creating forms for mobile devices is very much the same as designing other forms in Omnis. In Omnis Studio 5 you can now design multiple layouts for the same remote form, catering for desktop and mobile clients in one single form.

The new remote form property \$screensize can be set to Desktop monitors, as well as Portrait or Landscape for mobile screen sizes. The fields on your form can be arranged for the different layouts, but only one set of methods, for fields and the form itself, is required. When you are designing a remote form for mobile, the Windows Mobile titlebar and menubar are displayed in the design window to give an authentic look.

Screen Size and Orientation

There are many different screen sizes and resolutions for smartphones and other mobile devices. The new version of Omnis Studio will support the most common, namely 240x320 screens that run at 96dpi, and 480x640 screens that run at 192dpi (the latter uses a 240x320 logical size after scaling – in other words each application pixel is 4 device pixels).

There is a new remote form property, called \$screensize, that stores the current screensize and orientation of the screen and can be set to one of several values:

pathname	:PicsRemoteForm
screensize	kSSZwindowsMobile240x320Portrait
showascheckedout	kSSZDesktop
showgrid	kSSZwindowsMobile240x240
sizetogrid	kSSZwindowsMobile240x320Portrait
startfield	kSSZwindowsMobile240x320Landscape
superclass	kSSZwindowsMobile320x320
title	kSSZwindowsMobile480x800Portrait
top	kSSZwindowsMobile480x800Landscape
userinfo	
version	

- kSSZDesktop**
For display on a desktop PC (in a web browser)
- kSSZwindowsMobile240x240**
For mobile devices with screens 240 x 240 px at 96dpi
- kSSZwindowsMobile240x320Portrait | kSSZwindowsMobile240x320Landscape**
For mobile devices with screens 240 x 320 px at 96dpi, and 480 x 640 at 192dpi, in Portrait/ Landscape orientation
- kSSZwindowsMobile320x320**
For mobile devices with screens 320 x 320 px at 96dpi

- ❑ **kSSZwindowsMobile480x800Portrait | kSSZwindowsMobile480x800Landscape**
For mobile devices with screens 480 x 800 px at 96dpi in Portrait/ Landscape orientation

kSSZDesktop is for desktop (non-mobile) clients, that is, remote forms running in a desktop based web browser, as in previous versions of Omnis. The other sizes are for the Portrait and Landscape orientations of mobile devices with various screen sizes; the commonest screen size for smartphones is 240 x 320 pixels.

The remote form stores an array of values for the left, top, width and height values for the form itself, and all of its objects, *for each of the supported screen sizes*. When you add an object to the class (either at design time, or at runtime using the notation), the coordinates are stored for the current value of \$screensize. Obviously at this point, coordinates for other screen sizes are not set, so if you change \$screensize, the coordinates returned will be obtained as follows:

- ❑ If the screen size has an opposite orientation (portrait/landscape), and the values are set for that orientation, use those values.
- ❑ If not, use the coordinates for the first screen size in the order of the list above that has values set.

Similarly, if you change class object coordinates (either at design time or by using the notation), the values affected are for the current value of \$screensize.

When you open the form in design mode, the appearance of the design mode window depends on the current setting of \$screensize. So for kSSZDesktop, the remote form will be displayed as in previous versions of Omnis Studio. For the Mobile screen sizes, the design window is of a fixed size, showing you the space available on the device, together with the menu and title bar areas of the Windows Mobile device screen.

Form Width and height

When \$screensize is set to one of the mobile formats, changing \$width and \$height for the remote form changes the width and height of the client area of the design window (the area where you can place controls), rather than the size of the design window itself. The design window adds scrollbars to the client area when necessary, therefore you don't need to enable the \$horzscroll or \$vertscroll for the remote form. In addition, you can control whether the design window shows the Windows Mobile title bar, by setting the \$designshowmobiletitle property for the form, although you may want to show this for most types of mobile applications.

In most cases, you should set the \$width and \$height of the remote form to match the exact coordinates of each of the screen sizes you wish to support in your application, allowing for the Windows Mobile Title bar and Menu bar which are both 26 pixels each in height. Given these criteria for setting \$width and \$height, and assuming the title bar and menu bar are enabled, you can use the following coordinates for the supported screen sizes.

\$screensize / orientation	Settings for remote form	
	\$width	\$height
kSSZwindowsMobile240x320Portrait	240	268
kSSZwindowsMobile240x320Landscape	320	188
kSSZwindowsMobile240x240	240	188
kSSZwindowsMobile320x320	320	268
kSSZwindowsMobile480x800Portrait	480	748
kSSZwindowsMobile480x800Landscape	800	428

In addition, if you have set the \$effect property of the remote form to any effect other than kBorderNone, you must subtract a further 2 pixels from the above coordinates to allow for a 1 pixel border around the form.

When the client loads the remote form on the device, it takes the physical screen size of the device, and for rectangular screens it works out the coordinates for both portrait and landscape orientations. The client then looks at the width and the height of the form for each orientation. If the width and height of the form are different, the client will choose the appropriate form orientation. If the width and height of the form are the same, it treats the form as only having a single orientation, and the form will appear not to change when you flip the orientation of the mobile device.

Screen Events

When constructing a form, the Omnis Mobile Client uses the most appropriate screen size and orientation stored with the form, for the current screen size and orientation of the device. If the user swaps from portrait to landscape, or back again, the Omnis Mobile Client repositions the controls automatically, using the coordinates the new orientation stored in the remote form (assuming you have added a layout for each orientation). If for example, the user switches to landscape orientation, and you haven't added a layout for this orientation to the form, the portrait layout is used by default.

When the orientation changes, Omnis sends an evScreenOrientationChanged event to the top remote form. This allows the remote form to adjust the coordinates of any dynamically added objects. In addition, evFormToTop also receives the pScreenSize event parameter, allowing other forms to make adjustments if necessary when they come to the top.

If you want to use any screen events in your code, such as evScreenOrientationChanged, you have to enable the events in the \$events property of the remote form.

Testing Mobile Forms

Omnis Studio 5 allows you to test a remote form on your mobile device during development, by connecting your PC running Omnis and your mobile device using the Windows Mobile Device Center (ActiveSync on older platforms). Assuming your mobile device is setup correctly:

- Connect your mobile device and development PC using a USB cable or cradle

- Check that the device can use the PC's Internet connection, for example, by running Mobile IE on the device
- Click on Studio 5.x in the Studio Browser and click Prefs to open the Property Manager
- In the Omnis preferences, click on the \$webbrowser property and set it to 'Windows Mobile' by clicking the button; "wm:" will be added to the text box which is a shortcut for the Windows Mobile client
- Right-click on your remote form or press Ctrl-T to open your form on the mobile device

When you press Ctrl-T, Omnis checks to see if the client is present and up-to-date on your mobile device, and if not, it installs the client automatically. Omnis also checks individual components within the form and updates them if required via the Omnis Component Manager.

From there on, whenever you test the remote form from within the IDE, the form will open on your mobile device. You can change the form as much as you like and continue to test it using Ctrl-T.

Troubleshooting

If, for some reason, you have the Omnis Mobile Client (omwebcli) already running on your mobile device you may get an error when you try to open another instance. You need to open the Task Manager on your mobile device and stop the omwebcli application. To do this, tap on Start, then Settings, select the System tab, tap on the Task manager, check the Omnis Studio option and tap on Stop Selected. This applies to Windows Mobile 6.1; so the procedure for accessing the Task Manager for version 6.0 and 5.0 will be different.

Configuring the Mobile Client

The mobile forms you build in Omnis are run inside the Omnis Web Client, which itself runs inside a small stand-alone program, called **omniswm.exe**, installed on the mobile device and configured using an XML file. In effect, the new client program and XML file replace the ActiveX and HTML file required for web browser based Omnis applications and available in previous versions of Omnis. The XML configuration file specifies the Omnis library name, remote form name, Omnis Server location, and any other additional parameters.

The Omnis Mobile Client runs in two modes: *test form mode*, and normal *client mode*. In test form mode, the configuration is in the file **testformconfig.xml**; otherwise, it is in the file **config.xml**. There are two files to prevent test form from overwriting a configuration file you may want to keep: test form works by substituting placeholders in testformconfigtemplate.xml, copying the resulting file to testformconfig.xml on the client, and running the client.

The configuration files are located in the same folder as the Omnis Mobile Client (omniswm.exe), usually in the \Program Files\omwebcli folder of your mobile device. You can navigate to this folder using the Windows File Manager on your desktop PC.

The configuration file has the following syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<OmnisMobileConfig>
  <WindowClass name="TIGERLOGICCORP_OMNISWM"/>
  <TitleBar disable="no" text="Application Name"/>
  <MenuBar candefault="yes"/>
<!-- Message for when the application is waiting for a data
connection -->
  <WaitingForDataConnection message="Waiting for a data
connection"/>
  <Application name=" ApplicationName" desc="This is my app">
    <ServerType unicode="yes"/>
    <WebServer url="http://www.myserver.com"
script="/omnis_apache"/>
    <OmnisServer address="5920" library="MOBILE"
class="NewRemoteForm1"/>
    <!-- Add up to nine parameters for the application here -->
    <Parameter name="p1" value="value1"/>
    ...
    <Parameter name="p9" value="value9"/>
  </Application>
</OmnisMobileConfig>
```

- The **WindowClass** name attribute uniquely identifies the client program to the Windows Mobile operating system. This ensures that only one copy of the program with this class exists.
- The **disable** attribute of the TitleBar parameter can be set to yes, in order to remove the Windows Mobile title bar. The text attribute gives the application a name displayed on the device, which if omitted, defaults to “Omnis Studio”.
- The **candefault** attribute of the MenuBar parameter can be set to no if you do not want the client to provide a default menu on the Windows Mobile menu bar.
- The **Application** name is any name you want to give the Application entry in the configuration file, it can also have a **desc**.
- The ServerType **unicode** parameter should always be set to Yes.

- ❑ **Webserver** parameters
The **url** and **script** parameters specify the web server url and location of the Omnis web server plug-in
- ❑ The OmnisServer **address**, **library** and **class** parameters specify the port number or IP address:port of the Omnis Server, the name of your Omnis library, and the remote form class name; these parameters are the same as the Web Client plug-in properties as in previous versions of Omnis.

Note that the OmnisServer **address** attribute can be prefixed by an IP address, even when the WebServer url and script are empty (as they are for test form): this allows the client to connect directly to Omnis running on the PC when testing the form.

You can specify up to 9 parameters using Parameter tags to pass to the \$construct() method of the remote task inside the Omnis application.

Mobile Client Updates

The Omnis Mobile Client itself and the various components available in the mobile client are updated on the end user device automatically via the **Omnis Component Download Manager**, as in previous versions of Omnis Studio.

There is a new control platform in the Component Download Manager for Windows Mobile, which can load Windows Mobile components into the control manager data file when running on Win32 platforms. It detects whether a DLL is for desktop Windows or for Windows Mobile.

In addition to the client DLLs, you must also add the file omniswm.exe to the control manager data file.

The End User Experience

End users need to install the Omnis Mobile Client onto their mobile device by downloading and running a cab file you have created and containing all the necessary components and the mobile client itself (see the Deployment section). When the end user displays a form in your application, the most appropriate screen size and orientation is chosen for the device, and they can switch between portrait and landscape orientation automatically (assuming you have designed the form with multiple layouts).

Users can navigate the form using the standard left, right, up and down arrow keys on the device, or using the touchscreen or stylus, while data can be entered using the keypad. In addition, you can now add context menus to remote forms (using the new Remote menu class: see later in this manual), which appear on Windows Mobile when the user taps and holds on the form.

Mobile Client Design considerations

There are various issues regarding the font table, styles, themes, borders, and so on, that you should consider when designing remote forms or applications for the mobile devices.

Font table

Studio 5.0 libraries have a new font table #WMWFFONTS containing the fonts to be used for remote forms displayed in Windows Mobile; note that there is no Windows Mobile

report font table, since Omnis reports are not possible on mobile devices. The font table editor has a new column for the Windows Mobile fonts.

In addition, the Change Font Table dialog has changed, so that there is now a single grid, where each font cell uses a combo box to allow you to select a font:

Desktop platforms

For desktop platforms, the combo box list is either populated with the fonts available for the current platform, or a hard-coded list for other platforms (actually obtained from the resources used to initialize a new font table).

Windows Mobile

The combo box list for Windows Mobile is either hard-coded, or if possible, when running on Win32, Omnis will obtain the list of fonts for Windows Mobile from the device. This may result in Omnis attempting to install the client on the device: see the test form section for more details on client installation.

Style table

Studio 5.0 libraries have a new built-in style platform, `kWindowsMobile`, used for remote forms running on Windows Mobile. The initial values in `kWindowsMobile` are copied from `kWindows`.

Design DPI

The library property `$designdpi` now has a fourth entry, which is the design DPI to be used for Windows Mobile. This defaults to 96, and for most purposes you won't need to change this value. Note that the scaling for 192dpi devices is not related to this property, rather the Omnis Mobile Client itself does the scaling to 192dpi automatically.

Borders and Themes

Windows Mobile supports only simple borders for fields, and as a result so does the Omnis Mobile Client. Setting `$effect` to any value other than `kBorderNone` has no effect and results in a single pixel black border around fields when the form is displayed on the client.

Remote forms now support the `$backgroundtheme` property. This change applies to desktop clients as well as mobile clients.

Scrollbars

The Omnis Mobile Client adds scrollbars to its client area automatically, if they are required to access all of the form. You should therefore avoid enabling scrollbars (`$horzscroll` and `$vertscroll`) for the remote form.

Default context menu for lists

Lists in the `formflds` component group, such as the headed list, now have a default context menu allowing selection of lines when in multiple select mode.

Keyboard Interface

Windows Mobile devices typically have only a SIP (Software Input Panel) accessed by the central button in the menu bar, and a few navigation keys. Left, right, up and down arrow keys perform tabs if the current context of the form control is appropriate, e.g. pressing up

arrow when a list is at line one will perform a shift tab. In addition, the center button of the arrow keypad can be used to carry out selection actions, e.g. check a check box.

Control Appearance

Remote form fields and other controls generally have the correct appearance for Windows Mobile. This works in a similar manner to other platforms, where the theme or other property needs to be set to a specific value to obtain the standard system appearance.

However, the Formtbar component (tab bar with square tabs) has the standard Windows Mobile appearance.

Formflds

There is a new checkbox list control in the formflds component package. This is available on desktop as well as mobile platforms. The \$statecolumn property works in a similar way to the fat client checkbox list.

The headed list now supports \$showcolumnlines and \$linehtextra properties. This change applies on desktop as well as mobile platforms.

Icons on 192dpi devices

If you use a 16x16 icon from an Omnis icon file or #ICONS, when running on a 192dpi device with automatic scaling, the Omnis Mobile Client uses the 32x32 icon if there is an equivalent available. This provides a better result than stretching the 16x16 icon.

Hyplinks

The Hyplinks control cannot track while the mouse is over it on Windows Mobile, because there is no mouse pointer to move around. When you click on the link it highlights differently on Windows Mobile.

Unsupported Controls

The following controls are not available for Windows Mobile:

- Formhpic, Formroll, Formtran
because they are too reliant on mouse tracking when the mouse is not pressed
- Formport
because this is not useful on a mobile device
- Formpri
because there is no printing API
- Formqt3
because there is no QuickTime API

Client method execution

The *Quit Omnis* command quits the Mobile Client when executed in a client method. It has no effect in a desktop client.

Mobile Form Events

The `evFormToTop` event has a new event parameter, `pScreenSize`. This is a `kSSZ...` constant that identifies the current screen size and orientation in use.

In addition, there are three new remote form events.

- evScreenOrientationChanged**
This applies only to `omwebcli`, and it was described earlier. Event parameters are `pEventCode` and `pScreenSize`
- evOpenContextMenu** and **evExecuteContextMenu**
Events for the new Remote menu class, See the Remote Menus section below.

If you want to use any screen events in your code, such as `evScreenOrientationChanged`, you have to enable the events in the `$events` property of the remote form.

Deployment

The Omnis Mobile Client (`omwebcli`) is packaged as a CAB file and is installed on your mobile device automatically when you test your remote forms during development (note it is placed in the “\Program Files\omwebcli” folder for development). For deployment, you need to create your own CAB file containing the Omnis Mobile Client and your configuration files. End users need to download your CAB file and run it on their mobile device. The CAB file should install the Omnis Mobile Client and other files into a sub-folder of the “\Program Files\<your-app-name>” folder.

Windows Mobile Deployment tool

There is a new tool, called the Omnis Windows Mobile Deployment tool, available in the Web Client Tools section on the Add-on menu, for creating a CAB file for deploying your mobile application to end-user devices.



The Windows Mobile Deployment tool allows you to create a CAB file for your application, containing your own values for information such as the application name, icon, shortcut link in the Programs folder, and the connection configuration for your Omnis Server. The CAB file created by this tool installs the client application in “\Program Files” on the mobile device, in a folder named with the abbreviated name of your application.

The parameters that relate to the Omnis Server (e.g. Omnis Server, Omnis Library, etc.) are identical to the parameters you set in your XML config file for your Mobile Client application during development or testing. The following options can be set in the Windows Mobile Deployment window:

- Load Settings button**
The Load Settings button allows you to load the settings from a CAB file you have previously created with this tool.
- Company name**
Your company name, which the user will see on their device.
- Abbreviated app name**
the name for your mobile application used on the client; up to 8 characters
- Programs folder app name**
the name used for the Programs folder shortcut for your mobile application
- Program icon**
the Windows icon file (.ico) for your client application. The icon file must contain icons for 16x16, 32x32, 48x48 formats. If omitted the Omnis icon is used.
- Window class name**
The class name for the client application window. This must be unique for the entire Windows Mobile system on the client device. Therefore, you are advised to use a combination of your company and application name to create a unique class name. You cannot use TIGERLOGICCORP_OMNISWM as it is reserved.
- Application title**
The title used in the client application window.
- Show title bar**
Displays the title bar in your client application window.
- Allow default menu**
If checked, the right menu button will use the default Omnis menu if you do not supply your own right menu
- Unicode server**
Specifies the Omnis server should be Unicode compatible; leave this checked since all Studio 5 servers are Unicode compatible.
- Web server URL**
The URL or server location of the Web Server through which your mobile client application connects to the Omnis Server.
- Web server script**
The path or location of the Omnis Web Server plug-in, usually located in your cgi-bin or scripts folder.

- ❑ **Omnis Server**
The port number or combination of IP-address and port number (ipaddress:port) of the Omnis Server containing your Omnis mobile application.
- ❑ **Omnis Library**
The name of the Omnis library containing your Omnis mobile application.
- ❑ **Omnis class**
The name of the remote form in your Omnis mobile application; this can be the name of your application's initial remote form if you have more than one remote form.
- ❑ **Busy image DAT file**
The image to be used during communication between the client device and the server application. This can be created using the Image Compiler available under the Web Client Tools option in Omnis.
- ❑ **Waiting for data connection message**
The message for when the application is waiting for a data connection.
- ❑ **Parameters**
List of up to 9 parameters and their values that you can send to the \$construct() method in the remote task in your Omnis application.

Single Mobile Client

You should note that the Windows Mobile OS supports only a single version of the Omnis Mobile Client installed on a device. In practice, this is unlikely to cause you many problems since the majority of your end users will not have multiple versions of the Omnis Mobile Client on their mobile device. This is because the Windows Mobile OS only supports a single loaded DLL of each DLL module name over the entire system. This means that if you create a Windows Mobile based Omnis application installed in its own directory (e.g. using a CAB file created by our deployment mechanism), the application must use the identical Omnis Mobile Client to all other Windows Mobile based Omnis applications installed on the device.

The consequence of using two different versions are that the first executed will load and run, but the second may not, as it could be using a mixture of DLLs from the first and the second, because Windows Mobile uses any loaded DLLs with the same name from the first installation when it loads the second. One solution is for the user to terminate the process for the first application before running the second, but this is not practical for end users. The best solution is to ensure that the end user has only one version of the Omnis Mobile Client installed on their mobile device.

Server and Client Requirements

To deploy your mobile application, you need to install an Omnis Server, to host your Omnis library, database and other application files, and a standard Web Server. In this respect, the setup is very similar to previous versions of Omnis when deploying Web Client applications.

Your mobile application can be hosted on a Windows, Mac OS X, or Linux based Omnis Server even though clients are currently supported only on Windows Mobile based devices. You will also need to install the Omnis web server plug-in into the Web Server, but you

don't need to set up any html files to configure the client, since the XML configuration file configures the client.

To use your mobile application, your end users need to download and install the CAB file you created containing the Omnis Mobile Client and your own XML configuration files.

The Omnis Mobile Client is supported on Windows Mobile 'Professional' version 5.0, 6.0 and 6.1. The latest version 6.1 is typically available on the new breed of smartphones with touchscreens, such as the Sony Ericsson Xperia X1, Samsung Omnia, and the HTC 'Touch' range. The Omnis Mobile client does not work on the Standard version of Windows Mobile.

Direct Client Connections

In addition to the existing technique of connecting web based clients to an Omnis Server via a Web Server, the new Web Client plug-in (omwebcli) can connect directly to Omnis, without the need for a Web Server. This capability is available for the web browser based clients and new Windows Mobile based clients.

The new direct way of connecting web and mobile clients to the Omnis Server may be useful in certain circumstances, and enables you to test your web and mobile application without setting up a web server. It also allows you to develop a Windows Mobile client when running the Omnis Server on Mac OS X or Linux, without needing to set up a Web Server.

To enable direct connections to the Omnis Server you need to make some modifications to the html file for web based clients or the config.xml for mobile clients. The WebServer script attribute needs to be set to /webclient, and the WebServer url attribute needs to be set to http://<ipaddress>:port. In this case, the OmnisServer address attribute is not relevant when connecting directly to Omnis in this way.

Omnis Mobile Tutorial

To show how quickly and easily you can create a mobile form in Omnis Studio 5, this section uses the existing Tutorial library and shows you how to adapt a remote form for use on a Windows Mobile based device. If you don't have a mobile device, you can install a Windows Mobile 6 compatible emulator, available to download from the Microsoft website.

To try this short tutorial, you can either use the final version of the library, called Pics.lbs and provided in the Welcome\Tutorial folder, or work through some of the tutorial until you have created the appropriate database session, schema class and remote form.

Opening/Creating a Library

You can open the final version of the tutorial library, Pics.lbs, from the Welcome\Tutorial folder, which is located in your Local folder under Windows Vista at something like:

```
C:\Users\
```

The advantage of using the Pics.lbs library is that all the classes have been created for and the library opens a session to the Pics database ready for you to use.

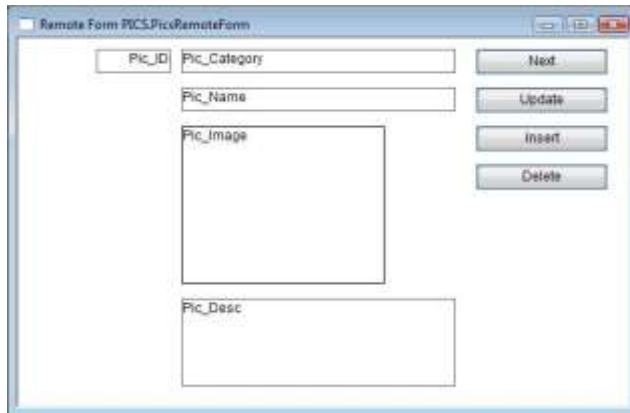
Alternatively, to create the library from scratch, you need to follow the tutorial in the 'Introducing Omnis' manual or in the Welcome screen (available when you start Omnis or by clicking on the New Users button), until after the section called 'Creating a web form' (page 37 to 39 in the manual). After that stage in the tutorial, you should have created the remote form called *PicsRemoteForm*. Note you can skip the sections on creating the desktop form (window class), menu class, and report class; you only need to create and open a database session in the SQL Browser, create the schema class, and create the remote form based on the Pics database using the SQL Form Wizard.

Adapting your Remote Form for Windows Mobile

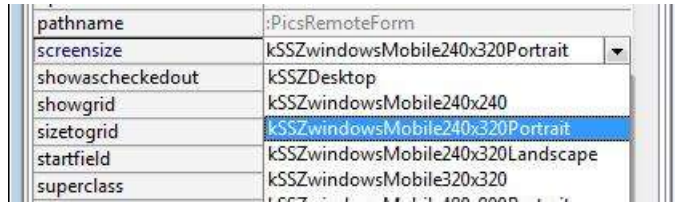
Assuming you have opened/created the *PicsRemoteForm* in the Pics tutorial library, and that you have opened the Pics database session (called PICSESS in the final tutorial library; note the session is opened for you in the final version of the Pics library):

- Open the remote form *PicsRemoteForm*
- Open the Property Manager and set the form background to white (in the final tutorial library the background is blue)
- Select all the fields (not the buttons) on the form, set their \$backcolor property to white
- Assign a simple plain border to all the fields by setting \$effect to kBorderPlain (note that kBorderPlain is the only effect applicable to Windows Mobile)

Your remote form should look something like this:



- Click on the background of the form and open the Property Manager (press F6)
- Under the General tab, click on the \$screensize property

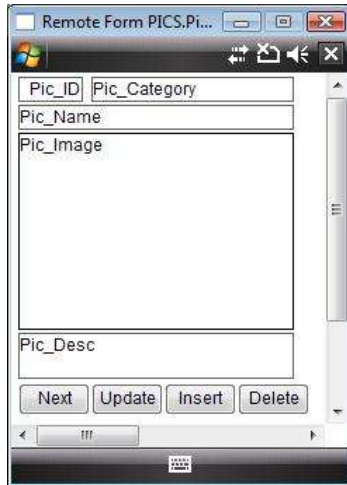


- Select kSSZwindowsMobile240x320Portrait from the \$screensize droplist

The remote form will resize to a format suitable for display on mobile devices, and in this case, in the portrait orientation.

- Resize and rearrange the fields and buttons on the remote form to fit the new mobile format (you'll need to use the scroll bars on the form to access all the fields/buttons)

Your form should look something like the following:



When you're happy with arrangement of the fields and buttons, you need to set the width and height of the form, which will also remove the scroll bars.

- Click on the background of the form and change \$width to 240, and change the \$height property to 268

Note that these values for \$width and \$height are for the portrait screen size / orientation only, that is, \$screensize = kSSZwindowsMobile240x320Portrait. The calculation for the \$height is 320 px, less 26 px each for the Windows Mobile title bar and menu bar. Later in this section you can setup the form for the Landscape orientation, which will have different values for the width and height.

Testing your Mobile Form

To test your form locally, between your desktop PC and your mobile device, you need to connect your device via a USB cable or cradle and set a few options in Omnis. You must have the Windows Mobile Device Center installed on your PC (this is called ActiveSync in Windows XP and older platforms).

- Connect your mobile device to the PC running Omnis Studio using a USB cable or cradle, and check that the device can use the PC's Internet connection (e.g. run Mobile IE by tapping Start >> Internet Explorer on your device)
- In Omnis, click on 'Studio 5.x' in the Studio Browser and click the Prefs option to display the Omnis preferences in the Property Manager



- Click on the \$webbrowser property in the Property Manager, and in the dialog that opens, click the 'Windows Mobile' button



- "wm:" will be added to the text box which is a shortcut for the default location of the Omnis Mobile Client (if the client is not on your device it will be installed automatically when you test the form for the first time)
- In the Studio Browser, right-click on the PicsRemoteForm and select Test Form, or if the remote form is on top, press Ctrl-T

If this is the first time you have used the Omnis Mobile Client, Omnis will try to install it on your mobile device. The message "Check your device for installation messages" should appear in Omnis on your desktop PC. The 'omwebcli.cab' should run on your device, installing the Omnis Mobile Client into the Program Files folder on your device.

- Confirm the installation on your mobile device, and in Omnis, confirm the installation was successful

The remote form should open on your device. You can tap the Next button and you will see data from the Pics database. If you don't see any data, check that the database is open in Omnis (in this case the Pics.df1 datafile) and that the PICSESS session is open and visible in the SQL Browser (the final Pics library should open the database session automatically).



From there on, whenever you test the remote form from within the Omnis Studio IDE, the form will open on your mobile device.

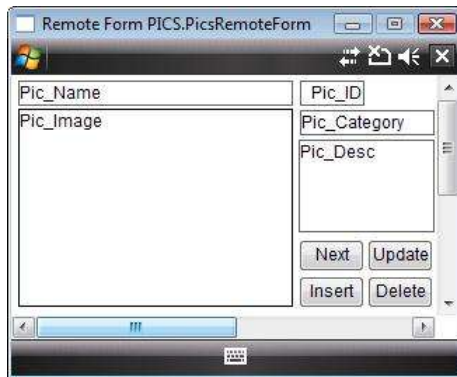
- Close the Omnis Mobile Client by tapping on the Close box on your mobile device

Creating a Landscape Form Orientation

You have designed your remote form for Portrait format, but you can add a different layout for devices capable of using the screen in landscape orientation: Omnis can store different layouts, for displaying your remote form on desktops and on various mobile screen formats, in the same remote form.

- Display the remote form `PicsRemoteForm` and open the Property Manager
- Change the `$screensize` property to `kSSZwindowsMobile240x320Landscape`

Omnis will change the orientation of the form in design mode and you can rearrange the fields and buttons to suit this format. Your form may look something like the following:



Next you need to set the width and height of the form for this orientation, which will also remove the scroll bars.

- Click on the background of the form and change the \$width property to 320, and change \$height to 188 (note the height of the Windows Mobile menu and title bar, 26 px each, is subtracted from the height of the form)

To test this layout, you can right-click the form and select Test Form, which will open the form on your mobile device. You need to change the orientation of your mobile device to landscape: some devices allow you to “flip up” the screen in a sideways direction to reveal a full Qwerty keyboard, while others respond to a sharp movement or reorientation of the device. When you change the orientation of the device, the remote form should change orientation. Again, you can tap the Next button and you will see data from the Pics database.



The different layouts for the form are stored in the remote class and Omnis uses the appropriate layout of fields and other controls depending on the current client and its orientation. The mobile client will “flip” from portrait to landscape automatically.

Remote Form Methods

You can examine the methods in the PicsRemoteForm remote form by double-clicking on the background of the form. Note that although the form you have created has different layouts for web clients (desktop browsers) and mobile devices, the form has *only one set of methods* for the form itself and all its fields. You can examine the \$construct() method for the remote form class (under Class methods), and the \$event() methods for each of the buttons: these methods work exactly the same for both web and mobile clients.

Client Configuration File

During development the Omnis Mobile Client is configured by the `testformconfig.xml` configuration file. You can examine the file by navigating to the `\Program Files\omwebcli` folder on your device and opening the file in Notepad or an XML editor (you can do this on your mobile device or on your desktop PC using the File Explorer). The file should look something like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<OmnisMobileConfig>
  <WindowClass name="TIGERLOGICCORP_OMNISWM"/>
  <!-- You can remove the title bar area by setting disable to
  "yes" -->
  <TitleBar disable="no"/>
  <!-- Normally you would use your own menus. If a menu is not
  specified and candefault is "yes" the client supplies a default
  menu -->
  <MenuBar candefault="yes"/>
  <Application name="PicsRemoteForm" desc="">
    <ServerType unicode="yes"/>
    <WebServer url="" script=""/>
    <OmnisServer address="192.132.60.111:6666" library="PICS"
    class="PicsRemoteForm"/>
  </Application>
</OmnisMobileConfig>
```

See earlier in this document for an explanation of the elements in this file.

Remote Menus

Omnis Studio 5 introduces a new class type, the Remote menu class, which can be added to remote forms and objects. From the user's point of view, the new remote menu class provides *context menus* for remote forms, which can be displayed in your web browser based applications and on mobile devices. In addition, you can use the new remote menu class to provide left and right menus exclusively for the menu bar on Windows Mobile based devices.

Creating Remote Menu Classes

The new remote menu is implemented using a different mechanism from existing menu classes and instances. The remote menu editor is very similar to the editor for existing menu classes, except that there is no access to methods, because the new remote menu class does not contain any methods. In addition, the properties of the new remote menu class and its objects are different from the existing desktop menu class.

You can associate a remote menu with a remote form, or a remote form control, by setting the `$contextmenu` property of the form or individual control. In addition, you can add

menus for the left and right menus of the Windows Mobile menu bar, by setting the `$menuname` property of the remote form. If you do not specify a menu for the right-hand menu, the client may supply its own default right menu, depending on the `MenuBar` setting in the configuration file for the Mobile Client. Objects have a `$disabledefaultcontextmenu` property. If true, the default context menu for the object will not be generated in response to a context click (`$lib.$disabledefaultcontextmenu` and `$obj.$disabledefaultcontextmenu` must both be false for the default menu to be generated). The default menu could be the clipboard menu for the edit control.

The key property of a remote menu line is `$commandid`. This allows you to execute the appropriate code when the menu line is selected; see the next section.

There is a new group, `$remotemenu` in the library object, containing all remote menu classes for the library. There is no `$remotemenu` group.

Context Menu Events

Context menus for remote forms work in a similar way to context menus for windows, that is, if an object is clicked on and it does not have a context menu, the client will display the context menu for the innermost enclosing container that has a context menu, or no context menu if none can be found. The default menu of an object is included in these checks.

After the user has Right-clicked on a form or object, and a context menu has been located, the client generates an `evAfter` for the current field, followed by an `evOpenContextMenu` event. This event is sent to the object that has the context menu property set. It has an event parameter, `pClickedField`, which is an item reference to the object on which the user has Right-clicked, as opposed to the field which has the context menu property set. It also has an event parameter, `pContextMenu`. This is an item reference to the remote menu instance for the menu that is about to open. You can also access this menu instance, using the notation `$inst.$remotemenu` (`$inst` refers to the remote form instance).

If you want to use any context menu events in your code, such as `evOpenContextMenu`, you have to enable the events in the `$events` property of the remote form.

The remote menu instance exists only during the `evOpenContextMenu` event. `evOpenContextMenu` can be processed on the client, or on the server. You use the remote menu instance to modify the menu before it is displayed on the client. One key property of a remote menu instance object is `$remotemenu`: for hierarchical menus, this is the item reference to the remote menu instance of the attached remote menu.

After `evOpenContextMenu` completes (note that you can discard the event to prevent the menu from being displayed), the client displays the menu:

- If the user dismisses the menu without selecting an item, no further event occurs.
- If the user selects a remote menu item, the client sends an `evExecuteContextMenu` event to the form or form control that received `evOpenContextMenu`, passing the event parameter `pCommandID` set to the value of `$commandid` for the selected menu line.

In the case of Windows Mobile, the mechanism is identical when the user opens one of the menus on the menu bar. In this case, the remote form receives the events, and `pClickedField` is a reference to the remote form. In addition, the `$order` property of the

remote menu instance allows you to determine which menu was used: it has the value kRMOContext, kRMOLeft or kRMORight.

Web Server Plug-in

The Omnis Web Server plug-in manages communications between Web and Mobile based clients and your Omnis application hosted on the Omnis Server. For Omnis Studio 5, you can configure the Web Server plug-in via a separate configuration file, allowing greater security and control over user access to your Omnis application. The parameters specified in your configuration file can provide default connections for clients, simplifying the post command required to connect to the Omnis Server. The new configuration file should be placed in the same directory as your Web Server plug-in.

There are three enhancements to the web server plug-in available on the Windows and Linux platform.

- ❑ **Access and Security**

You can restrict access to an Omnis Server via the use of a configuration file.

- ❑ **Configure Server parameters**

The configuration file allows you to override parameters for the Omnis Server, or in effect, provide default parameters if the OmnisServer parameter is blank in your cgi parameters; in this case the value would be taken from the configuration file.

- ❑ **Post content to Remote tasks**

The final enhancement is the ability to pass HTTP Post content to the Omnis remote task, again using the configuration file.

Server plug-in activation

The new functionality in the Omnis Web Server plug-in is built into the plug-ins supplied with the Omnis Server installation, but you have to rename plug-in itself to activate the new functionality.

mod_omnis.so

If you are using mod_omnis.so under Linux or Windows, you need to change the value of the location in your http.conf or equivalent apache configuration script to /omnis_apacheini, for example:

```
<location /omnis_apacheini>  
    SetHandler omnis-apache  
</location>
```

nph-omniscgi

Rename the nph-omniscgi.exe to nph-omniscgiini.exe for Windows, or rename nph-omniscgi to nph-omniscgiini for Linux.

omnisapi.dll

For Windows IIS based servers, rename omnisapi.dll to omnisapiini.dll.

rdtaserv.dll

If you are using the Web Services enabled Web Server plug-in, rename rdtaserver.dll to rdtaserverini.dll.

Creating a Configuration file

The configuration file should be named omnissrv.ini and be placed in the same directory as your Web Server plug-in, for both Windows and Linux.

The format of the configuration file mirrors that of a Windows .ini file and is defined as follows:

- Section names are contained in square brackets e.g. [SectionName].
- A section ends when another section begins or at End Of File (EOF).
- Comments are lines beginning with a semicolon (;).
- All text following a comment is ignored until the line is terminated.
- Keys are of the form keyname=value where keyname is a unique identifier within the section and value is the value of the specified key.
- Section names, key names and key values must not contain white space.
- Section names and key names are case sensitive.

The new functionality in the Web Server plug-in is controlled using specific named sections in the configuration file. The omnissrv.ini file can contain the AllowConnectionsTo section which controls access to the Omnis Server. The .ini file can also include either a DefaultConnection or OverrideConnection section (but not both), which either provide default parameters for the Omnis Server or override parameters posted to the Omnis Server from the http web server.

Controlling Server Access

You can control access to your Omnis Server by including the [AllowConnectionsTo] section in the configuration file. This section contains a list of key names of the form *address<n>* where *n* is a sequentially numbered character starting at 1. When this section is present, connections to Omnis Servers are limited to those defined in the specified key values. In the event that the OmnisServer parameter is defined as a port number, only the port number is required.

For example, in the following AllowConnectionsTo section connections are limited to Omnis servers running on the local machine on port 5920 and the remote machine 192.168.0.2 on port 5920.

```
[AllowConnectionsTo]
address1=5920
address2=192.168.0.2:5920
```

Note that the local IP address in the configuration file cannot be resolved. Imagine the server plug-in and the Omnis server are on the same machine, with an IP address of 192.168.0.3. If the incoming request was of the form OmnisServer=5920, the configuration file has to match this form. So if you want to allow only connections to port 5920, you

would have to add this line to the [AllowConnectionsTo] section: address1=5920. If you use the expanded form of the address, i.e. address=192.168.0.3:5920, the server plug-in would deny the request. In the event of a denial of service the plug-in returns a HTTP 403 error with the following message 'Access to the resource has been denied'.

Default Connections

You can provide default connection parameters in the [DefaultConnection] section of the configuration file. This section provides a means of adding missing values into an HTTP post, or in effect, providing a complete set of default parameters if none are provided in the HTTP post. When they are present in the HTTP request, the values in DefaultConnection are ignored and the values are taken from the original request. The DefaultConnection section can contain the following keys:

- OmnisServer
- OmnisClass
- OmnisLibrary
- PostDataParamName
- Any number of additional parameter pairs in the form *Parameter Name=value*

The OmnisServer, OmnisClass and OmnisLibrary mirror the operation of the identically named remote form parameters. The value of the PostDataParamName key specifies a variable name for all the content of the HTTP post. All other keys are assumed to be parameters. They are passed to the Omnis remote task and appear as columns in the row variable. The column name is the key name and the value matches the value of the key. One thing to note, if the parameter is present in the original request and the configuration file also contains a definition for the parameter, the value is always taken from the request even if the parameter has no associated value. For example:

```
[DefaultConnection]
OmnisServer=192.168.0.1:5920
OmnisClass=remoteTask
OmnisLibrary=TEST
param1=value1
param2=value2
PostDataParamName=PostData
```

In the context of the above DefaultConnection section, consider the following URL which attempts to connect to Omnis:

```
/omnis_apacheini?OmnisClass=remoteTask&param1=1234
```

The OmnisClass and param1 values are taken from the URL while the other values are taken from the DefaultConnection section. In this case, no OmnisServer and OmnisLibrary parameters are provided in the query string, so those values are taken from the configuration file. Therefore the plug-in will amend the query string to:

```
/omnis_apacheini?OmnisClass=remoteTask&OmnisServer=192.168.0.1:5920&
OmnisLibrary=TEST&param1=1234&param2=value2&PostData=
```

Note PostData is empty as the content-type is application/x-www-form-urlencoded, so in this case the data is not passed to Omnis.

Overriding Connections

You can override the server parameters passed to the Omnis Server by an HTTP post by including a [OverrideConnection] section in your configuration file. In this case, all the values in the request are ignored, and the Omnis Server uses values from the configuration file. The OverrideConnection section may contain the following keys with associated values:

- OmnisServer
- OmnisClass
- OmnisLibrary
- PostDataParamName
- Any number of additional *Parameter Name=value*

These keys function exactly as described in the DefaultConnection section. An example OverrideConnection section is as follows:

```
[OverrideConnection]
OmnisServer=192.168.0.1:5920
OmnisClass=rtTest
OmnisLibrary=TEST
param1=value1
param2=value2
PostDataParamName=PostData
```

In the context of the above OverrideConnection section, consider the following URL which attempts to connect to Omnis:

```
/omnis_apacheini?OmnisClass=remoteTask&param1=1234
```

In this case, the values in OmnisClass and param1 submitted in the post are ignored, and all the values for the post are taken from the DefaultConnection section in the configuration file. Therefore the query string is amended to:

```
/omnis_apacheini?OmnisClass=rtTest&OmnisServer=192.168.0.1:5920&
OmnisLibrary=TEST&param1=value1&param2=value2&PostData=
```

Note PostData is empty as the content-type is application/x-www-form-urlencoded, so in this case the data is not passed to Omnis.

Unicode

Omnis Studio 5.0 fully supports Unicode, which means you will be able to expand the market for your Omnis applications by supporting the majority of world languages and the display of special characters, including scientific and mathematical symbols.

In previous versions of Omnis Studio, we provided a Unicode and non-Unicode version of the development kit, but Omnis Studio 5 is Unicode compatible by default. The Unicode version of Omnis Studio is available for Windows, Mac OS, and Linux, and will allow you to localize your applications and deploy them to virtually any market, anywhere in the world.

What is Unicode?

Unicode provides a mechanism for representing characters or symbols used in many of the languages in the world, as well as scientific and technical environments. The Unicode standard is maintained by the Unicode Consortium (www.unicode.org) who set the standards for Unicode and promote its worldwide use. They define Unicode as: *“a character coding system designed to support the worldwide interchange, processing, and display of the written texts of the diverse languages and technical disciplines of the modern world.”* In the context of client-server and multi-tier computing, Unicode allows the seamless exchange and processing of data across different platforms, software products and programming environments.

The Unicode consortium provides information and resources concerning Unicode, including the standard definition and maintenance, character code tables, a locale identifier repository, and lists of Unicode enabled products. The last major version update of Unicode was version 5.1 which is capable of representing over 100,000 different characters, used in many different languages throughout the world. Many operating systems and software products have adopted Unicode, which is now universally accepted as the standard for character representation. For example, the latest versions of Windows Vista and Mac OS X, as well as all varieties of Linux, offer Unicode support. All web standards, such as the latest versions of HTML and XML support Unicode, as well as the latest versions of Internet Explorer and all Mozilla-based browsers. In addition, the most recent versions of Sybase, Oracle, and DB2 all provide Unicode support.

Together with the display of multiple languages in Omnis, the use of Unicode encoding affects the sort order of dynamic data, for example, in list variables, as well as the querying and retrieval of data from Unicode compatible Server databases.

Omnis Data File Conversion

WARNING: YOU SHOULD MAKE A SECURE BACKUP OF YOUR OMNIS DATA FILES BEFORE CONVERTING THEM IN THE UNICODE VERSION OF OMNIS STUDIO.

When you access an Omnis data file you are asked to confirm that you want to convert the data. After you select Yes, Omnis displays a dialog which offers two types of conversion:

- Full**
whereby a full conversion of the Character based data takes place and the indexes are rebuilt
- Quick**
whereby the indexes are dropped and rebuilt, but the data is not converted. This is OK for files containing only 7 bit data: Omnis does not check that the file contains only 7 bit data, so it's your responsibility to know whether or not it is safe to run this conversion process.

The full data file conversion mechanism converts the data in your Omnis data file and rebuilds the indexes. When data file conversion takes place, all data marked as Character is converted, including any characters ≥ 128 . In the case where character data is stored in a binary or external file, for example, text stored in a document file, conversion of this data *does not* take place.

Testing Data File Conversion

Omnis Studio 5 can perform a full conversion of Omnis data files to Unicode, as described above. We suggest that, having made a secure back up, you convert your Omnis data files to Unicode using the 'Full' conversion mechanism, and report any problems to us as soon as possible, so that any issues can be resolved before the final release version of Omnis Studio 5.0.

In particular, please check the results of full conversion carefully, and do not discard your backup of the non-Unicode data file until you are satisfied that the data has converted successfully. You could perform some regression tests on your application and data – you should normally do this with a new version of Studio, but when converting to Unicode Omnis, and converting your data files, you need to be especially sensitive to possible data file and indexing issues.

Data File commands

The *Open data file* and *Prompt for data file* commands have an existing option called "Convert without user prompts". If this is checked, and the new "Full Unicode conversion" option is checked, no dialogs are displayed and your data is converted to Unicode using the Full conversion process.

DAMs

The DAMs provided with Studio 5.0 are able to function in Unicode or 8-bit compatibility mode. This means that after converting your existing libraries for use with Studio 5.0, it should be possible to continue interacting with non-Unicode databases.

In 8-bit compatibility mode, all DAMs:

- Return non-Unicode character data types via the \$createnames() and \$coltext attributes
- Bind outgoing character variables using the database's non-Unicode data types
- Convert all data inside outgoing character bind variables to single-byte characters
- Define incoming character columns using the database's non-Unicode data types
- Convert all data inside incoming character bind variables from bytes into the Omnis character set

Switching to 8-bit compatibility mode

To switch to 8-bit compatibility mode, there is a session property \$unicode which should be set to kFalse from its default value of kTrue. This implementation allows multiple Unicode and 8-bit session objects to exist side by side if required.

Character Mapping

This section is applicable to session objects operating in 8-bit compatibility mode only.

When reading data from a server database, Omnis expects the character set to be the same as that used in an Omnis data file. The Omnis character set is based on the Mac OS extended character set, but is standard ASCII up to character code 127. Beyond this value, the data could be in any number of different formats depending on the client software that was used to enter the data.

When assigned, the \$mactable session property identifies files containing translation tables for 8-bit character codes read into and sent out of Omnis. For example, suppose you are working with a database that stores EBCDIC characters. In order to accommodate this database, you should create an '.IN' map file that translates EBCDIC characters to ASCII characters when Omnis is reading server data and a matching '.OUT' file that reverses the process by converting ASCII to EBCDIC characters when Omnis is sending data to the server.

Under Windows and Linux, Omnis uses the same character set as under Mac OS, so in the general case, mixed platform Omnis applications should have no need for character mapping. However, if the data in a server table was created by another software package, running under Windows for example, the characters past ASCII code 127 would appear incorrect when read using Omnis. In this situation the \$mactable property should be used to map the character set.

There are two kinds of character maps: IN and OUT files. IN files are used to translate characters coming from a server database into Omnis. OUT files are used to translate characters that travel from Omnis back to a server database.

The Character Map Editor

The Character map editor is accessed via the Add-On tools menu item and enables you to create character-mapping files. You can change a given character to another character by entering a numeric code for a new character. The column for the Server Character for both .IN and .OUT files may not actually represent what the character is on the server. This column is only provided as a guide. The Numeric value is the true representation in all cases.

To change a character, select a line in the list box and change the numeric code in the Server Code edit box. Once the change has been recorded, press the Update button to update the character map. You can increase/decrease the value in the Server Code edit box by pressing the button with the left and right arrows. Pressing the left arrow decreases the value, pressing the right arrow increases the value.

The File menu lets you create new character map files, save, save as, and so on. The Make Inverse Map option creates the inverse of the current map, that is, it creates an ".IN" file if the current file is an ".OUT" character map, and vice versa.

Using the Map Files

Establish the character mapping tables by setting the session property \$mactable to the path of the two map files. Both files must have the same name but with the extensions .IN and .OUT and be located in the same folder. The \$mactable property establishes both .IN and .OUT files at the same time. For example:

```
Do SessObj.$mactable.$assign('C:\Program Files\Tiger Logic\
  Charmaps\pubs') Returns #F
```

In this example, the two map files are called "pubs.in" and "pubs.out".

The session property \$charmap controls the mode of character mapping that is to be applied to the data. Set the character mapping mode using a command of the form:

```
Do SessObj.$charmap.$assign(pCharMap) Returns #F
```

The potential values for the character mapping mode parameter pCharMap are:

- kSessionCharMapOmnis**
Use the internal Omnis character set.
- kSessionCharMapNative**
This is the default and specifies that the client machine character set is to be used.
- kSessionCharMapTable**
Use the character mapping table specified in the \$mactable property. If the \$mactable property is not set and the application attempts to assign kSessionCharMapTable this fails.

If you wish to use the character mapping tables defined using the \$mactable property, you must set \$charmap to kSessionCharMapTable.

Interpreting 8-bit Data

This section is applicable to the MySQL, PostgreSQL and Openbase DAMs which interface with their respective client libraries using the UTF-8 encoding.

When operating in Unicode mode, it is possible to receive mixed 8-bit and Unicode data—since UTF-8 character codes 0x00 to 0x7F are identical to ASCII character codes.

Where this data was created using the non-Unicode version of Omnis however, it is possible that the data may contain ASCII extended characters. In this case, the Unicode DAM will encounter decoding errors, mistaking the extended characters as UTF-8 encoded bytes.

This issue was not a concern for the non-Unicode version of Omnis Studio since extended characters were always read and written as bytes, irrespective of the database encoding.

In order to avoid problems when upgrading to the Unicode version of Omnis Studio, it is advisable to convert tables containing ASCII extended characters to UTF-8. This process is simplified where the database character set is already set to UTF-8 (as is often the case with MySQL). All that is required is to read and update each row in the table and repeat this for all tables used by the application. In so doing, Omnis will convert the 8-bit data to Unicode and then write the converted Unicode data back to the database.

In order to facilitate this within the DAM, the session property `$validateutf8` is provided. When set to `kTrue` (the default), any fetched character data is validated using the rules for UTF-8 encoding. Where a given text buffer fails validation, it is assumed to be non-Unicode data and is interpreted accordingly. When written back to the database, all character data will be converted to UTF-8. Such updates will result in frequently accessed records having their contents refreshed automatically.

By setting `$validateutf8` to `kFalse`, validation is skipped and the DAM reverts to the previous behavior, in which case extended ASCII characters should be avoided.

Aside from the issue of UTF-8 encoded data, the DAMs provided with Studio 5.0 are able to retrieve non-Unicode data from non-Unicode database columns in either Unicode or 8-bit compatibility mode. The DAM knows the text capabilities of each character data type and assigns encoding values to each result column accordingly.

The difference in behavior when using 8-bit compatibility is that in compatibility mode, it is also possible to write data back to non-Unicode columns.

In Unicode mode, the DAM assumes that it will be writing to Unicode compatible data types and this will cause data insertion/encoding mismatch errors if the clientware tries to insert into non-Unicode database columns.

Character Mapping in Unicode Mode

Character mapping to and from the Omnis character set is also possible where session objects are operating in Unicode mode. This was previously removed from the Unicode DAMs since it provided compatibility between the various 8-bit character sets. Where Unicode DAMs encounter 8-bit data however, it is necessary to indicate the character set used by the data. For this reason the session `$charmap` property can be used to indicate that fetched 8-bit data uses either:

- `kSessionCharMapRoman`**
Use the Mac Roman character set to interpret the characters
- `kSessionCharMapLatin1`**
Use the Windows/Linux character set to interpret the characters

Fetching Data to a File

The `$fetchtofile()` method has the `iEncoding` parameter, as follows:

```
Do StatementObj.$fetchtofile(cFilename [,iRowCount=1]
    [,bAppend=kTrue] [,bColumnNames=kTrue]
    [,iEncoding=kUniTypeUTF8/kUniTypeLatin1])
    returns Long integer
```

where `iEncoding` is an optional parameter specifying the type of encoding to be used. It should be one of the Unicode type constants and defaults to `kUniTypeUTF8`. The corresponding Unicode Byte Order Marker (BOM) is written to the beginning of the file when the file is empty or when `bAppend` is set to `kFalse`.

Server Specific Programming

Certain DAMs, namely DAMORA8 and DAMODBC also provide session properties which allow mixing of Unicode and 8-bit data when the DAM is operating in Unicode mode.

Oracle DAM

This section summarizes recent changes made to the Unicode Oracle Object DAM designed to enable insertion and retrieval of mixed ANSI and Unicode character types.

In the case of Oracle 8i and later, these data types are:

CHAR	Fixed single-byte character data, limited to 2000 bytes.
NCHAR	Fixed multi-byte character data, limited to 2000 bytes. (1000 UCS-2 encoded characters)
VARCHAR2	Varying length, single-byte character data, limited to 4000 bytes.
NVARCHAR2	Varying length, multi-byte character data, limited to 4000 bytes. (2000 UCS-2 encoded characters)
CLOB	Character Large Object- single-byte character data.
NCLOB	National Character Large Object- multi-byte character data.
LONG	Varying length, single-byte character data, limited to 2GB. Supported for backward compatibility only.

By default, the Unicode Oracle DAM maps all Omnis character data to the `NVARCHAR2` and `NCLOB` data types, dependent on the field length of the Omnis bind variable.

However, the Oracle DAM provides session properties which affect the Omnis to Oracle data type mappings:

`$nationaltonvarchar`

If set to `kTrue`, Character and National data types are treated differently when being inserted to `VARCHAR2` / `NVARCHAR2` columns. The *National* character subtype will be used with Unicode data, whilst the *Character* subtype will be reserved for non-Unicode data.

❑ **\$nationaltonclob**

If set to kTrue, large Character and National data types are treated differently when being inserted to CLOB / NCLOB columns. The onus is upon the developer not to put Unicode characters into Character subtypes when using these properties; otherwise data insertion/encoding mismatch errors will occur.

❑ **\$maxvarchar2**

Sets the byte limit above which Omnis character fields will be mapped to CLOB/NCLOB data types as opposed to VARCHAR2 / NVARCHAR2 columns. The maximum value is 4000 bytes.

❑ **\$longchartoclob**

If set to kTrue (the default), Omnis large character fields > \$maxvarchar2 in byte length will be mapped to the CLOB/NCLOB data type. If set to kFalse, the LONG data type is used.

Reading Unicode and Non-Unicode Data

The Oracle DAM automatically detects the data type of retrieved character columns and converts the data accordingly. There is no need to modify any properties in order to retrieve mixed ANSI and/or Unicode Data.

ODBC DAM

The ODBC DAM provides the \$nationaltowchar session property.

By default, Omnis Character and National fields are mapped to the SQL_WCHAR, SQL_WVARCHAR and SQL_WLONGVARCHAR data types. By setting \$nationaltowchar to kTrue only National fields will be mapped to these types (to the equivalent server data types) and Character fields will be mapped to SQL_CHAR, SQL_VARCHAR and SQL_LONGVARCHAR as determined by the Omnis field length. Character fields mapped in this way are subject to data loss/truncation where such fields contain Unicode characters. When setting this property, please note that Unicode data types usually have precision limits half that of their corresponding ANSI data types. For example, this is 8000 for the SQL Server VARCHAR() data type but 4000 for NVARCHAR(). \$nationaltowchar affects both the text returned by the \$createnames() method and the binding of input parameters.

Character Normalization

Originally, Unicode was a 16-bit character set. It has subsequently been extended to include code point values up to and including U+10FFFF. It is not expected that it will be extended any further. Windows and Mac OS X still represent Unicode character strings using arrays of Short (16-bit) integers. This is not a problem, because the UTF-16 standard allows code points U+10000 and greater to be represented by pairs of 16-bit values (each member of the pair occupies space in the 16-bit range that is not used for code points). This representation is referred to as a *surrogate pair*.

Internally Omnis uses UTF-32 to represent code points, that is, each code point occupies 32 bits, and the value of each code point is between 0 and U+10FFFF inclusive. This allows for straightforward processing of character strings, since every code point occupies the same space in memory.

Unicode allows a significant number of characters to be represented by more than one sequence of code points. For example, consider the letter E with circumflex and dot below, a character that occurs in Vietnamese (Ệ). This character has five possible representations in Unicode:

1. U+0045 Latin capital letter E
U+0302 combining circumflex accent
U+0323 combining dot below
2. U+0045 Latin capital letter E
U+0323 combining dot below
U+0302 combining circumflex accent
3. U+00CA Latin capital letter E with circumflex
U+0323 combining dot below
4. U+1EB8 Latin capital letter E with dot below
U+0302 combining circumflex accent
5. U+1EC6 Latin capital letter E with circumflex and dot below

A character represented by more than one code point is referred to as a *composite character*. A character represented by a single code point is referred to as a *pre-composed character*.

As far as the end-user is concerned each of these representations usually needs to be treated identically. This leads to some interesting consequences for Omnis. These are discussed in the following sections. Note the term *end-user character* means the character that the end-user is working with – in the example above, the end-user character is Ệ.

Normalization of a Unicode character string converts the string into a standard, defined format. Once normalized, a Unicode character string has only one possible representation, thereby making it possible to compare it with other character strings, and produce results useful to the end-user. The Unicode standard recommends two forms of normalization.

These are:

1. Canonical decomposition, referred to as NFD:
Pre-composed characters are replaced by their equivalent composite characters;
Composite characters are replaced with a single fixed composite representation.
2. Canonical decomposition followed by canonical composition, referred to as NFC:
After carrying out NFD, all composite characters are replaced with their pre-composed equivalent, where one exists.

Omnis provides two functions to normalize character strings:

- ❑ **nfd(string)** carries out canonical decomposition on the string and returns the normalized string.
- ❑ **nfc(string)** carries out canonical decomposition followed by canonical composition on the string and returns the normalized string.

These functions are **not** available in client-side web client methods.

Comparing Text

Omnis uses two types of comparison for character strings:

- ❑ Comparison of the UTF-8 values of the strings. This is called Character comparison.
- ❑ Comparison according to the rules for the locale specified via the localization data file; prior to comparison, the input data is normalized. This is called National comparison. National comparison is more likely to produce results that the end-user would expect. Note that upper casing used in conjunction with national comparison may not have an effect, since sometimes the rules for the locale ignore the case of the characters.

The `natcmp()` function uses national comparison. Note that `natcmp()` is **not** available in client-side web client methods.

Omnis compares text for many different reasons, and in many different places. Key areas are:

- ❑ Sorting lists
- ❑ Searching lists
- ❑ Manipulating data file indexes
- ❑ Expressions, for example the test on an if statement

Omnis supports two types of character variable – character and national.

Sorting Lists

When using the character type, Omnis uses character comparison.

When using the national type, Omnis uses national comparison.

Searching Lists

When using the character type, Omnis uses character comparison.

Searches that directly use a character column of national type use national comparison.

Other searches, for example searches using a calculation, will behave as if they are operating on normal character data. However, you can use `natcmp()` as part of the calculation, in order to use national comparison.

Manipulating Data File Indexes

Indexes for national fields use national comparison.

Expressions

To ensure the correct behaviour of expressions that test the value of character variables, you must either normalize their value first using `nfc()` or `nfd()`, or you must use the `natcmp()` function.

Drawing Text

Depending on the font and operating system you use, different representations of the same end-user character may not always be drawn in the same way. The same applies if you try to use strings that require surrogate pairs. Generally speaking, you will get the best results if you normalize the text using `nfc()`, as the issues generally occur with composite characters.

Entering Text

Wherever possible, you are recommended to use the `nfc()` normalization form for data that is to be edited. If composite characters are present in the data, multiple left or right arrow key presses are required to skip a composite character, and also clicking and selecting in the text will highlight an area which when copied to the clipboard might not exactly contain what appeared to be highlighted.

Omnis performs NFC normalization on character data pasted from the clipboard when running in the thick client (runtime); no normalization occurs when pasting characters into a remote form when using the web client.

Character Translation

The following functions have been added to allow you to translate a specified character in a string to its Unicode value and to allow the reverse.

- ❑ `unicode(string,position[,returnhex])`
returns the Unicode value of the character at the specified position in the string. The first position in string is 1. If Boolean `returnhex` is true (default false) it returns a hex string representing the value, of the form 'U+h'.
- ❑ `unichr(num1[,num2]...)`
returns a string formed by concatenating the supplied Unicode character codes. Each code is either a number or a string of the form 'U+h', where h is 1-6 characters representing a hexadecimal value.

These functions are available in client-side methods as well as the thick client, but will generate an error if used in the non-Unicode version of Omnis.

Unicode Clients

Locale Identifier

A new function `locale()` returns the Locale Identifier (LCID) for the current client machine/operating system. As well as the language of the machine, the Locale Identifier specifies the decimal, thousand and list separators, currency values, units of measurement, date formats, and character sort order. The Locale is specified at the operating system level and is in the form `language_country`, where *language* is the ISO639 language name, and *country* is the ISO3166 country name. For example, the Locale for the UK is "en_GB". On Mac OS X, there may be other information, such as a script code, between the language and country (this is because OS X uses ICU locales).

Note that `locale()` does not work in methods executing in the web client under Mac OS X.

Unicode Data Handling

The `unicodev()` function allows you to translate Unicode character data from one type to another. The syntax is:

```
unicodev(srcType, src, dstType, dst, bom, errtext)
```

The function converts `src`, and stores the result in `dst`. It returns zero for success, or a non-zero error code together with error text in `errtext`. `Src` and `dst` are either binary or character variables, depending on the values of the `srcType` and `dstType`.

srcType and **dstType** are one of the `kUniType...` constants (see below).

Bom is Boolean: if true, **dst** has a Unicode Byte Order Marker (BOM) if relevant for the destination type.

The `kUniType...` constants are as follows:

- ❑ **kUniTypeAnsi...**
The data is stored in a binary variable, and contains character data where each byte is encoded using the specified ANSI code page. A range of constants are provided to cater for most world or regional languages, including Cyrillic, Greek, Hebrew, Arabic, Thai, and so on
- ❑ **kUniTypeAuto**
The source encoding is automatically detected from the conversion source; possible encodings are identified by the remaining `kUniType...` constants (allowed only for the source type).
- ❑ **kUniTypeCharacter**
The data is stored in a character variable. Note – this constant has been moved since the last Unicode build, so you need to re-enter it in your code.
- ❑ **kUniTypeNativeCharacters**
The data is stored in a binary variable and contains a stream of bytes, where each byte is a character in the Latin 1 character set for the machine (Ansi on Windows, MacRoman on the Mac, ISO-8859-1 on Unix)
- ❑ **kUniTypeOEM**
Each byte in the data represents a character in the OEM character set
- ❑ **kUniTypeUTF16**
The data is stored in a binary variable and contains Unicode character data encoded using UTF-16LE if the machine is little-endian, or UTF-16BE if the machine is big-endian. Useful when writing cross-platform code that interacts with the OS.
- ❑ **kUniTypeUTF16BE** or **kUniTypeUTF16LE**
The data is stored in a binary variable and contains Unicode character data encoded using UTF-16BE (big-endian) or UTF-16LE (little-endian)
- ❑ **kUniTypeUTF32**
The data is stored in a binary variable and contains Unicode character data encoded using UTF-32LE if the machine is little-endian, or UTF-32BE if the machine is big-endian. Useful when writing cross-platform code that interacts with the OS.

- ❑ **kUniTypeUTF32BE or kUniTypeUTF32LE**
The data is stored in a binary variable and contains Unicode character data encoded using UTF-32BE (big-endian) or UTF-32LE (little-endian)
- ❑ **kUniTypeUTF8**
The data is stored in a binary variable and contains Unicode character data encoded using UTF-8

Formfile

The \$filereadencoding and \$filewriteencoding properties have been changed. In previous versions of Omnis Studio, the Formfile component defined kFFEncoding... constants. These constants should not now be used, and you are encouraged to use the kUniType... constants to identify the file encoding. Formfile has been extended, so that you can use any of the kUniType... constants except kUniTypeCharacter for the \$filereadencoding property, and any of the kUniType... constants except kUniTypeAuto and kUniTypeCharacter for the \$filewriteencoding property.

In addition, there is also a kUniTypeBinary constant to identify files that are to be treated as raw binary data.

Code that uses the old kFFEncoding... constants should continue to work.

Fileops

The Fileops component has two methods, \$readcharacter() and \$writecharacter() (introduced in a previous version of Omnis), which allow you to read and write Unicode character data from and to a file.

- ❑ \$readcharacter(encoding,variable)
reads all data from a file containing character data into *variable*; *encoding* is one of the kUniType... constants (listed above), identifying the encoding of the file.
- ❑ \$writecharacter(encoding,variable)
replaces the contents of the file with the character data stored in *variable*; *encoding* is one of the kUniType... constants, identifying the encoding of the file.

For \$readcharacter, specify the encoding as any kUniType... constant except kUniTypeBinary and kUniTypeCharacter.

For \$writecharacter, specify the encoding as any kUniType... constant except kUniTypeAuto, kUniTypeBinary and kUniTypeCharacter.

Note the \$readcharacter() and \$writecharacter() methods use the kUniType... constants and not the kFFEncoding... constants which should not now be used.

Mixing Char & Binary data

You cannot concatenate a Character variable to a Binary in the Unicode version of Omnis Studio. The correct method is to use \$readfile to read the file into a Binary variable, and then parse the binary variable. Assigning Character to Binary and vice-versa is likely to cause problems, including data corruption, and should therefore be avoided.

Import/Export and Report File Encoding

There are a number of properties that control the encoding of import text files, export files, and report data written to text files and the port. These are:

- ❑ **\$importencoding**
The encoding used for imported data when importing from port, or when the import file does not have a Unicode Byte Order Marker (BOM). Any of the kUniType... constants, except kUniTypeAuto, kUniTypeCharacter, kUniTypeBinary and the kUniTypeUTF32... values.
- ❑ **\$exportencoding**
The encoding used for exporting data and printing to port or text file. Any of the kUniType... constants, except kUniTypeAuto, kUniTypeCharacter and kUniTypeBinary.
- ❑ **\$exportbom**
If true, and the \$exportencoding preference identifies a Unicode encoding, a Unicode BOM is output at the start of the output file.

These properties can be found in the Omnis preferences (\$root.\$prefs). In a multi-threaded server, there is a separate value of each of these properties for each thread.

Localization

Support for localization has been significantly enhanced in Omnis Studio 5 which will allow you to provide better support for multiple languages in your Omnis applications. Specifically, Unicode support in Omnis means that the number of foreign languages and character sets you can use in your Omnis applications is greatly expanded. The tools for the localization of your libraries and the Omnis executable itself (omnisloc.lbs) have also been improved in this version.

The use of the String Label object and String Tables allow you to provide alternative languages for text labels, and other text objects in your Omnis libraries. The String Label and support for String Tables are now available for remote forms to allow you to localize applications running in the Web Client.

In addition to the ability to change text in your libraries using String Tables, Studio 5 introduces two new string tables to allow you to translate a greater range of built-in resources in the Omnis executable and Omnis Web Client, such as built-in messages, and standard menus and toolbars.

Note for existing developers

The Translation Library (called trans.lbs) has been removed and is no longer in the Local folder under the main Omnis folder. You should use the String Label object and String Tables to support multiple languages in your Omnis libraries.

Also note that the information regarding String Labels and the String Table Editor, that was provided in the localib.pdf document in previous versions, is reproduced here for developers who are new to the localization process.

Localizing Your Libraries

Omnis contains an external package called ‘StringTable’ that contains a special String label object and a set of functions that allow you to dynamically change the language of text labels in your Omnis application.

You can use the String Table Editor in Omnis to create tables containing a matrix of strings or words for any number of languages. String tables and the functions in the StringTable package allow you to load the text for fields, buttons, and text labels when the window is opened, and then change the language of the text while the window is open.

The String Table Editor allows you to translate a whole list of strings (stored in the first column of your string table) into one or more languages automatically. For example, you could add English labels to a window or remote form, add the text for these labels to a string table, and with a single click translate all the labels to French, German, and Italian, with a single mouse click. The String Table Editor uses the translation tools provided by Google Translate™ to translate the text in your string tables automatically.

Using String Labels

First you need to create your window or form in your application that you wish to display in multiple languages. If you are enabling an existing window for dynamic label and text translation, you will need to replace existing text labels with the special ‘String Label’ external component from the ‘StringTable’ package. You can use standard fields and buttons.

Locating the StringLabel component

- ❑ **For window classes**
the StringLabel object is located in the ‘Background Components’ group in the Component Store. The StringTable component is pre-loaded for window classes so you don’t need to load it by right-clicking on the Component Store.
- ❑ **For remote forms**
the StringLabel object is located in the ‘WEB Background Objects’ group in the Component Store. If the object is not shown in the Component Store, right-click the Component Store, select the ‘External Components...’ option, and load the String Label object (FORMSTRG) from the ‘Form Background Components’ node in the list.

To create String labels

- Open your window or remote form and open the Component Store (press F3)
- Click on the ‘Background Components’ or ‘WEB Background Objects’ group in the Component Store toolbar and drag the String Label component onto your window or remote form
- Open the Property Manager (F6/Cmnd-6), click on the Custom tab, and enter a suitable label in the \$rowid property

The row ID can be a number or string, so you could use Label_01, Label_02, etc, or something like Label_Firstname, Label_Lastname, etc, or whatever convention you want.

- Create the other labels for your window or form using the String Label component, and assign unique labels in the \$rowid property for each one

It is also possible to translate the text for pushbuttons, check boxes and the contents of lists that may appear in your window or form. To identify these objects in the string table and your Omnis code you should make sure the objects have a suitable text string specified in the \$name property. The text in the \$name property of an object should be used as the row ID in the string table and also used in your code to reference the text. You could use names such as Button_01, Button_02, etc, or something like Button_Next, Button_Update, etc. or whatever convention you want.

Editing String Tables

Having created the fields and string labels in your window, you need to set up a String Table that contains all the alternative text for each object in the different languages you wish to support.

Note The first column of a string table was labeled “ID” but in Studio 5 is now called “STRINGID”. This is because “ID” is the ISO 639 code for Indonesia, and the Studio 5.0 localization features use ISO 639 codes to identify the language columns, so “ID” could no longer be used. Old string tables, where column one is called “ID” will usually still work, unless of course you are using ID to represent Indonesia in one of the language columns.

To create a string table

- Open the String Table Editor from the Tools>>Add-Ons menu
- Click on New to clear the table and create a new file; you can click on Save to save the new string table; the file should have the .stb file extension and can be located in the same folder as your library

The first column in the new string table is called STRINGID: do not rename this column since this will contain the object rowIDs used to identify the strings in the table.

- Rename the second column by clicking in the column and selecting the Rename Column option from the Columns menu; name the column using the two-letter ISO 639 language code for your “native” or default language, e.g. enter EN if your string labels and buttons contain English
- Next you need to add a row for each string label or button you have used in your window(s) and/or remote form(s) in your application, entering the row IDs and button names you have used to identify these objects in the STRINGID column

You can tab to the end of the row to create a new row or click the Add Row button and enter the Row ID for the next table entry; continue adding rows for each label or object you wish to translate.

- Add a column for each language you wish to support in your application, using the two-letter ISO 639 language code to name each column, e.g. the third column could be named FR and the fourth column could be named DE to support French and German

- Next click on the Translate button, select the ‘Translate from’ column and the ‘Translate to’ columns; note the ‘Translate to’ list allows multiple selections to allow you to translate to multiple languages, or you can press Ctrl/Cmnd-A to select all lines (except the one for the ‘Translate from’ column)
- Click the Translate button and Omnis will translate all the strings; note the time taken to translate all your strings will depend on how many rows and columns are in your string table

The following example shows a String Table for a remote form that has a number of string labels and pushbuttons. The English strings are in the second column, while the French and German strings were added automatically using the Translate option.



STRINGID	EN	FR	DE
Button_Delete	Delete	Supprimer	Löschen
Button_Insert	Insert	Insérer	Legen Sie
Button_Name	Name	Nom	Name
Button_Update	Update	Mettre à jour	Aktualisieren
Label_Category	Category	Catégorie	Kategorie
Label_Desc	Description	Description	Beschreibung
Label_ID	ID	ID	ID
Label_Image	Image	Image	Image
Label_Name	Name	Nom	Name

- Then you need to save and close the string table
- Finally, click on the background of your window or form, open the Property Manager (F6), and set the \$stringtabledata property to the name of your string table; you can click on the property dropdown and navigate to your string table

Accessing String Tables via the Catalog

String Tables can be accessed and edited via the String Table tab in the Catalog (F9) window. Your own string tables for windows and remote forms will appear in the Catalog window when the window or form is the top design class.

Omnis Studio 5 has two new string tables that allow you to translate various built-in string resources in the Omnis executable, and the string resources in the Web Client. These are visible in the Catalog and can be edited in the String Table Editor from there.

String Table Functions

The StringTable package contains a number of functions that allow you to load string tables, load text items from a table, and replace the text in the windows in your application. The following methods are available:

- ❑ **\$colcnt()**
StringTable.\$colcnt([cTableName]) returns the number of columns in string table cTableName, or an error code which is less than zero
- ❑ **\$getcolumnname()**
StringTable.\$getcolumnname([cTableName]) returns the current column name for the string table specified by cTableName, or an error code which is less than zero
- ❑ **\$getcolumnnumber()**
StringTable.\$getcolumnnumber([cTableName]) returns the current column number for the string table specified by cTableName, or an error code which is less than zero
- ❑ **\$gettablelist()**
StringTable.\$gettablelist(IList) populates a single column IList with the loaded string table names; define IList to have a single character column before calling this method
- ❑ **\$gettext()**
StringTable.\$gettext(cRowID) returns the text from the cell specified by cRowID for the current column, or an error code which is less than zero
- ❑ **\$loadcolumn()**
StringTable.\$loadcolumn(cColumnNumber|Name,cTableName,cPathname) loads column cColumnNumber|Name from string table at cPathname into table cTableName. Returns kStringTableOK or an error code which is less than zero
- ❑ **\$loadexistingtablefromlist()**
StringTable.\$loadexistingtablefromlist(cTableName,IList) replaces an existing string table with the content of a list. Returns kStringTableOK or an error code which is less than zero
- ❑ **\$loadlistfromtable()**
StringTable.\$loadstringtable(cTableName,cPathname) loads string table from file cPathname, and gives it the name cTableName. Returns kStringTableOK or an error code which is less than zero
- ❑ **\$loadstringtable()**
StringTable.\$loadstringtable(cTableName,cPathname) loads string table from file cPathname, and gives it the name cTableName. Returns kStringTableOK or an error code which is less than zero
- ❑ **\$loadtablefromlist()**
StringTable.\$loadtablefromlist(cTableName,cPathname,IList) creates a string table from a list. Returns kStringTableOK or an error code which is less than zero
- ❑ **\$removestringtable()**
StringTable.\$removestringtable(cPathname) deletes the string table file specified by cPathname. Returns kStringTableOK or an error code which is less than zero

- ❑ **\$rowcnt()**
StringTable.\$rowcnt([cTableName]) returns the number of rows in string table cTableName, or an error code which is less than zero
- ❑ **\$savestringtable()**
StringTable.\$savestringtable(cTableName) saves the string table specified by cTableName. Returns kStringTableOK or an error code which is less than zero
- ❑ **\$setcolumn()**
StringTable.\$setcolumn(cColumnNameOrName) sets the current column. Returns kStringTableOK or an error code which is less than zero
- ❑ **\$unloadall()**
unloads all string tables from memory.
- ❑ **\$unloadstringtable()**
StringTable.\$unloadstringtable(cTableName) unloads the string table cTableName from memory. Returns kStringTableOK or an error code which is less than zero

The following functions apply to string tables for remote forms and are available for execution in client methods only:

- ❑ **stgettext()**
StringTable.stgettext(id) returns the string with the specified *id* from a string table, or empty if the lookup fails. *id* can be prefixed with 'TABLENAME.', or must be an id for the string table of the current form.
- ❑ **stgetcol()**
StringTable.stgetcol(table) returns the name of the current column for string table *table*
- ❑ **stsetcol()**
StringTable.stsetcol(table,col) sets the current column for lookups from string table *table* to the column with name col and returns Boolean true for success.

Programming String Tables

The following method loads a string table with the name 'Lang.stb' and sets the column containing the English text as the current column.

```
; custom method $loadStringTable
Calculate lvpath as sys(10)
Do
  FileOps.$splitpathname(lvpath,lvdrive,lvdirname,lvfilename,lvfile
  ext)
Calculate lvpath as con(lvdrive,lvdirname,"Lang.stb")
Do StringTable.$loadStringTable(iTableName,lvpath) Returns lnum
If lnum<>kStringTableOK
  OK message Error (Icon) {The Language String Table "Lang.stb"
  could not be loaded.}
  Quit method lnum
End If
; Select the English Language
Do StringTable.$setColumn("English") Returns lnum
Quit method lnum
```

Having loaded the string table and specified the current column, the following method can be used to load the text or string values into the appropriate field labels, buttons, and lists in a data entry window. Note that the IDs for each object are stored in custom constants that are defined in the the `Startup_task` in the library.

```
; custom method $loadFields
; Define the button and group box descriptions using $getText
Do StringTable.$getText(kPrintButton) Returns lval
Do $cwind.$objs.PrintButton.$text.$assign(lval)
Do StringTable.$getText(kLanguageButton) Returns lval
Do $cwind.$objs.LanguageButton.$text.$assign(lval)
Do StringTable.$getText(kEmpTitle) Returns lval
Do $cwind.$objs.stEntry_1022.$text.$assign(lval)
```

The next section of the method defines the dropdown lists for Sex (male/female) and Marital Status by loading the relevant entries from the current string table.

```

Set current list iSex
Define list {iField}
Do StringTable.$getText(kMale) Returns iField
Add line to list
Do StringTable.$getText(kFemale) Returns iField
Add line to list
Calculate iSex.$line as 1
Set current list iStatus
Define list {iField}
Do StringTable.$getText(kMarried) Returns iField
Add line to list
Do StringTable.$getText(kSingle) Returns iField
Add line to list
Calculate iStatus.$line as 1

```

Within your application you need to provide some way for the user to select and change the language setting. This could be done using a separate window containing a list of available languages and a button to set the selected language. The following method gets all language column names from the Lang String Table and puts them into an Omnis list.

```

; custom method $loadList
Set current list iLang
Define list {iLangField}
; Save the current column number
Do StringTable.$getColumnNumber() Returns lnum
; Build a list of all column names in the String Table.
Do StringTable.$colCnt() Returns maxcol
; Start from column 2 as column 1 is reserved for String Table IDs
For lcount from 2 to maxcol step 1
  Do StringTable.$setcolumn(lcount)
  Do StringTable.$getcolumnname() Returns iLangField
  Add line to list
End For
Do StringTable.$setColumn(lnum)
Calculate iLang.$line as pLine

```

When the user selects a language from the list, they should click a button with the following method which changes the language of the text on itself and the data entry window.

```
On evClick      ;; Event Parameters - pRow( Itemreference )
  Set current list iLang
  Calculate lval as ((iLang.$line)+1)
  ; need to offset by 1 since coll in string table has the rowID
  Do StringTable.$setColumn(lval)
  Do StringTable.$getText(kLangTitle) Returns lval
  Do $cwind.$objs.stLang_1016.$text.$assign(lval)
  Do StringTable.$getText(kSetLanguage) Returns lval
  Do $cwind.$objs.SetLang.$text.$assign(lval)
  Do $root.$iwindows.stEntry.$loadFields
  ; this line runs the $loadFields method again (see above) to
  change the objects in the data entry window
  Do method $loadList (iLang.$line)
  ; this line reloads the language list in the new language
  Do method $translateList
  Do method $redrawAll
```

The following method translates the language list depending on the Language selected. The code uses the String Table column headings to look up corresponding IDs from within the table itself.

```
; custom method $translateList
Set current list iLang
Calculate lnum as iLang.$line
Calculate lrowcnt as iLang.$linecount
For lcount from 1 to lrowcnt step 1
  Do StringTable.$getText(iLang.[lcount].iLangField)
    Returns iLang.[lcount].iLangField
  Calculate iLang.$line as lcount
End For
Calculate iLang.$line as lnum
```

Finally you need a method to redraw all the fields and text labels on any open windows or forms.

```
; custom method $redrawAll
Do $iwindows.$first() Returns ref
While ref
  Do StringTable.$redraw(ref.$hwnd)
  Do $iwindows.$next(ref) Returns ref
End While
```

Localizing Remote Forms

The String Label object is now available for remote forms, and together with string tables, allows you to localize your applications running in the Omnis Web Client. See the previous section for details about how to create string labels and string tables; the technique is the same for remote forms.

In Omnis Studio 5, remote form classes have a new property called `$stringtabledata`. The contents of this property is the data, in list format, from a standard Omnis string table. To populate `$stringtabledata` in the IDE, you can click on the droplist button on the property in the Property Manager, and select a string table file. When you press OK, Omnis takes a copy of the string table data and stores it in the class. If you cancel the dialog, Omnis asks if you wish to clear the data from the class, which is a convenient way to clear the contents of `$stringtabledata`.

When the web client creates an instance of the remote form class, on the client, it automatically loads the string table data as a client-side string table, and sets the string table name to the remote form class name.

The column names in the string table must be:

- `STRINGID` for column 1 – this is the standard for string tables, and the name cannot be changed.
- A two character ISO 639 language code for the second and subsequent columns; for example, `en` for English, `de` for German, `fr` for French.

When the client loads a new string table (because a remote form is being instantiated in some way), it uses the operating system locale of the client to locate the string table column to use for lookups. You can override this behavior by setting the remote task instance property `$stringtablelocale` to a two character ISO 639 language code to use instead of the client operating system locale, but you can only do this in `$construct` of the remote task. If there is no column in the string table for the desired locale (specified in `$stringtablelocale`), lookups default to using column 2.

The `$rowid` property of a remote form string label object can refer to one of the Web Client string tables. If `$rowid` is not prefixed with a table name (remote form name), then the string must be in the string table for the remote form containing the object.

Any character string property of a remote form control can be specified as `$st.id` (note that `$st` is not notation; it is just a special prefix recognized by the client). This tells the client to lookup `id` in the string tables for the client, and set the property value to the result of the lookup. The `$rowid` rules regarding the presence of a table name prefix also apply in this case. Note that when you get a property set in this way, the result is the result of the lookup, not `$st.id`.

There is a new remote form property called `$stringtabledesignform` which allows you to access the string table in the Catalog (F9) windows while designing the remote form.

There are three new “Client String Table” functions, available for execution in client methods only:

- `stgettext(id)`
returns the string with the specified *id* from a string table, or empty if the lookup fails.

id can be prefixed with 'TABLENAME.', or must be an id for the string table of the current form.

- ❑ `stgetcol(table)`
returns the name of the current column for string table *table*
- ❑ `stsetcol(table,col)`
sets the current column for lookups from string table *table* to the column with name *col* and returns Boolean true for success.

There is also a new column, "ClientLocale", in the row variable parameter passed to `$construct()` of the remote task. This contains the locale for the client, as returned by the `locale()` function; the first two characters are the ISO 639 language code of the client.

Localizing Omnis

In order to provide a completely foreign version of your Omnis application, you may need to translate various resources that appear in Omnis itself (rather than strings that appear in your libraries, as described in the previous section). This applies equally to applications that run on the desktop using the Omnis executable (Runtime), and applications that run on the Internet using the Omnis Web Client: you can translate string resources in the Omnis runtime and in the web client.

Omnis Studio 5 introduces two new String Tables that allow you to translate:

- ❑ Various built-in string resources in the Omnis executable, and
- ❑ String resources in the Web Client.

The new String Tables can be accessed via the String Table tab in the Catalog (F9) window and are edited using the String Table editor.

In addition, the Localization Library (`omnisloc.lbs`) is still available in the Local folder for you to translate further string resources in the Omnis executable.

Localizing Built-in Client Resources

The built-in resources comprise the string resources, and the strings in dialogs, in the Omnis core, externals and external components. Omnis has two new string tables used for the translation of built-in and web client resources. The new tables are located in the Local folder of the main Omnis tree, and are:

- ❑ **studio.stb**
containing translations of text strings for the fat client, comprising dialogs, components, and so on
- ❑ **client.stb**
containing translations of text strings for the Omnis Web Client

These string tables can be edited using the string table editor. However, there are some special rules regarding their structure: the `STRINGID` column is the exact value of the built-in resource text and cannot be changed; the other columns contain the translations of the built-in resource text, and they are named using a 2 character ISO 639 language code, in lower case.

Usually, there is no column for “en” (English), because the built-in resources are English. If you are using a mixture of languages in the built-in resources, then you can add a column for “en”. As a result, you will have cases where the STRINGID and language column for the built-in resource would have the same value. To avoid duplicating the value in the language column, you can enter “*” for that column, meaning the string does not need translating from its STRINGID.

When reading a string resource, Omnis looks up the string value in the string table, and if present, it replaces the string value with the translation. If no translation is available, the string remains unchanged. The string value lookup is case-sensitive, allowing the translation to contain the correct case for the translated text.

Omnis loads the studio.stb string table when it starts up, and when `$root.$prefs.$language` changes, and the appropriate language version of the resources is loaded. The column used for translations is the ISO 639 language component of the locale stored in the current language record. This is also stored in a small text file (locale.txt) in the Local folder of the Omnis tree, to cater for the fact that the string table is loaded sooner in the life of Studio than the localization data.

When a web-based client connects, Omnis sends client.stb to the client; this is handled via the cache, so it will not always be sent. The client loads client.stb, and uses the column corresponding to the client locale to obtain translations (the client locale can be specifically set using `$cinst.$stringtablelocale` for the remote task instance, or it can be allowed to default to the locale of the client machine).

Editing the Built-in Client Resources

You can open and edit the studio.stb and client.stb string tables via the Omnis Catalog (F9) window. You can click on the String Table tab in the Catalog, and right-click on the string table you wish to edit.

Localization Library

The Localization Library (omnisloc.lbs) allows you to translate the visible text, labels and messages in the Omnis executable itself; these can appear in the Runtime when you deploy your application, such as Yes/No messages, days of the week, and thousand separator characters.

In the Localization Library, the sort order field is now labeled Locale. There is a new check box below the Locale field, “Use Locale For Defaulted Items”. This determines the locale used for language items that have been left empty, so that they get a default value from the system:

1. If the Use Locale check box is checked, default values come from the locale stored in the language record.
2. If it is not checked, default values come from the operating system default locale on the client machine.

Local Language

The locale() function for the fat client now has an optional Boolean parameter, which when passed as `kTrue`, causes it to return the locale field value for the current `$root.$prefs.$language`.

Omnis VCS

The Omnis VCS is widely used by developers to manage their Omnis development projects and with each new release it has gained many new enhancements. Omnis Studio 5 includes various enhancements in the Omnis VCS including the ability to create branches in a project, the Find Class option, the ability to strip Comments from builds, the Checked out classes warning, and Project Property updates.

VCS Repositories

You cannot use existing VCS repositories in Omnis Studio 5. There have been a number of changes to the structure of the VCS which means repositories created in previous versions of Omnis will not work with Omnis Studio 5. The majority of changes were made to provide support for Unicode compatible repositories.

Existing VCS repositories must be re-created to ensure that they are in the new Unicode compatible format. Unfortunately, you will lose all the revisions, but you could keep the old repository and view your projects in an old version of Omnis Studio.

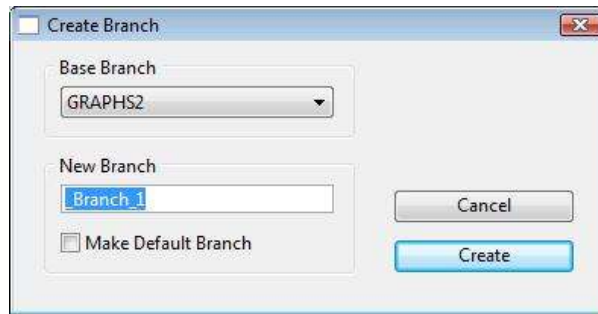
Project Branching

There is a new feature in the Omnis VCS that allows you to create 'branches' within the current project, based on the contents of your original project. You are then able to make changes or additions to the branches in your project, thereby in effect making different versions of the same project/application. Therefore, the ability to create branches may be of real benefit to developers who have different customers with differing or urgent requirements, outside of your normal product release cycle. In this case a branch can be created and tailored to an individual customer, perhaps to provide them with an urgent patch.

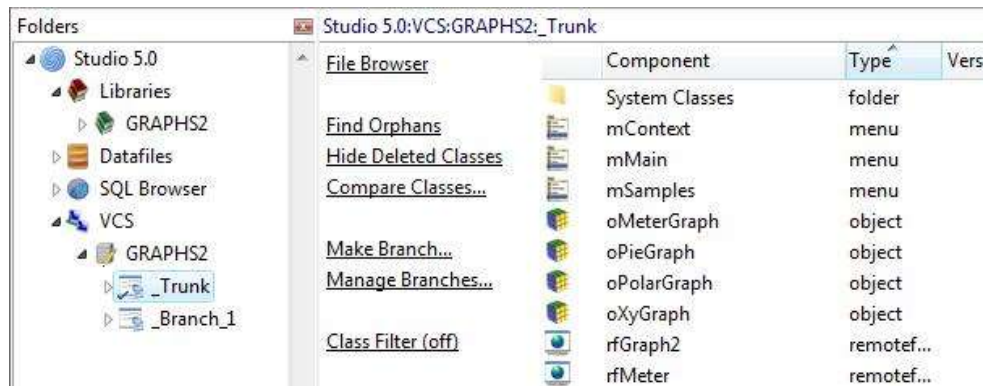
Project branching allows you to create and maintain multiple development paths alongside each other within a single project. Each branch is based on the original project branch, but is independent of each other. At present however, there is no function to merge different branches, for example, at a later stage in your project development cycle, so you should think carefully before creating multiple branches in a project.

Creating a Branch

You can create a branch by clicking the 'Make Branch' hyperlink in the Omnis VCS browser window. By default the new branch will be called '`_Branch_1`', but you can rename it either at this point or after creation using the Branch Manager tool. When you create a new branch, you can set it to be the default branch, although you'll probably want to keep the original branch as the default: see below.



The VCS creates the new branch and places the contents of your original project in another branch called ‘_Trunk’. At this point the new branch will be identical to the contents of ‘_Trunk’. Any assigned user privileges in the original project are carried over to the new branch. The VCS displays the two new branches or nodes underneath the original project name.



There is no difference between the ‘_Trunk’ and any other branch, that is, all branches have the same properties, so the Trunk simply identifies the contents of your original project. You can create as many branches as you wish, attached to the main Trunk, but you cannot create a sub-branch, that is, a branch of a branch.

The Trunk and Default Branch

When you create the first branch and you don’t set it to be default branch, the original development path in your project is named _Trunk and it becomes the default branch. A check mark icon indicates the default branch. The default branch can be changed at any time by right-clicking on a branch, but in most cases the original _Trunk can remain the default.

The following functions work only for the default branch:

- Build Updated Libraries
- Update from VCS
- Auto Checkin

Branch Options

There are two new options that allow you to manage the Checkin/Out process for projects that contain branches. The new options are under the Branches tab in the VCS Options window:

Use Default Branch

If unchecked and the project has branches, a list of available branches will be displayed to choose which one you want to Checkout from / Check in to when using the Class Browser context menus. This option is checked by default.

Ignore branches

If checked, the VCS automatically uses the branch the component has been checked out from (if there is more than one branch in the project, the Branch Choice window is displayed). Note: If you do not have this option checked, the VCS will populate the Branch Choice window with all the branches for this project and you have to select the branch to be used. This option is checked by default.

Building Projects with Branches

When a project is branched, the Build is only available at a branch level except for Scheduled Build, which will allow you to set the branch you wish to build at the specified time.

Managing Branches

The Manage Branches window lists all the branches defined for a project and allows you to label or delete branches. You can open the window from the Manage Branches hyperlink in the VCS browser.

Deleting a Branch

You can delete any branch apart from the `_Trunk`. To delete a branch you can use the Manage Branches window or Delete Branch hyperlink. If all branches are deleted from a project, the `_Trunk` is also removed, and all classes belonging to the `_Trunk` are reassigned to the original project.

VCS Reports

The reports in the VCS have been updated where appropriate to reflect the introduction of branching in projects.

An Example Scenario

Consider the following scenario: An Omnis developer releases a version of their application and begins work on the next version, checking changes into the VCS as the project progresses. Then a bug is identified in the released version which requires an urgent fix, perhaps for a particular customer who cannot wait until the next release to receive the fix. In this case, the developer could add a branch to the project, make the fix in the new branch and release an interim fix to satisfy the customer.

After the interim release is made, further work may be carried out on the “main product” branch (probably the Trunk/Default branch), which will then be released at some time in the future. In this case, the main product will not include the urgent fix, since it was made

to the other branch in the project. If the developer wants to include the urgent fix in the main product, they will have to add the fix to the main branch (or indeed multiple branches) to ensure that the fix is carried forward. Apart from this possible need to reconcile changes made to different branches, the ability to create branches may be of real benefit to developers who have different customers with differing or urgent requirements, outside of their normal product release cycle.

NOTE: please note there is no function at present to merge different branches, so you should think very carefully before creating multiple branches in a project and allowing separate independent development paths to be created.

Project Admin

Finding Classes

There is a new option in the VCS browser called “Find Class” that allows you to search for a class within the current VCS repository.

The new option is accessible from the hyperlink options at the VCS level in the Studio Browser. You can search for a class within one or more projects. A list of matching classes is displayed, and you can double-click a class to display that class in the appropriate project.

Checking Out

Checked Out Classes

There is a new option in the main VCS browser called “Checked Out Classes” that lists all the classes that are checked out by the current user, or for the Supervisor user the list shows the checked out classes for all users.

Updating Properties

When checking out a class, you can now update the destination library properties with the properties that are held in the VCS for that project. This is especially useful for situations in which there are multiple developers working on the same project. You can enable this new option, called “Update the destination library with preferences from the VCS”, under the Check Out tab in the VCS Options window.

Superclass and Design Task Names

There is a new option on the Check Out tab of the VCS Options to allow you to maintain the superclass or design task library name for checked out classes.

If checked, the superclass (or design task) library name is *not* removed from checked out classes. This is necessary when the superclass exists in a separate library and there is a class with the same name as the superclass in the current library.

Build Options

Removing Comments

When you create a build from your Omnis VCS repository, you can now remove the comments from the code in your libraries. There is a checkbox on the Build window to enable this option. If the option is set, all comments in code, variables & class definitions are removed.

Under the Build tab in the VCS Options window, there are four checkboxes that allow you to restrict this process from variables, file classes, query classes and schema classes.

Checked out Classes Warning

When you create a build from your Omnis VCS repository, the project may contain classes or objects that are checked out of the current repository. If you proceed with the build, your project may not contain the most up-to-date classes.

There is a new option in the VCS Options window under the Build tab called “Warn if there are classes checked out”. If set, the VCS will warn you that the build contains classes that are checked out. In this case, a window will open listing all the checked out classes with an option to proceed with the build or cancel it.

Time Settings

The Omnis VCS now supports UTC (Co-Ordinated Universal Time).

There is a new preference to set the local time zone, which you can set on the Display tab under the VCS Options. The VCS stores times in UTC, but displays times according to the time zone that has been set.

Methods

Method Performance

This version of Omnis Studio allows you to collect data about the performance of method execution in your application.

Collecting Performance Data

There is a new Omnis root preference called `$collectperformancedata` that enables method performance data collection. Its property is a `kCPD...` constant specifying whether or not and how Omnis will collect data about method execution performance. The data collected is stored with each method in its class, and can be accessed using the notation. Data is not collected for remote form client methods. The `kCPD...` constants are:

- kCPDnone**
Omnis does not collect method execution performance data
- kCPDallMethods**
Omnis collects method execution performance data for all methods

kCPDmarkedClasses

Omnis collects method execution performance data for methods in classes where the class property `$collectperformancedata` is `kTrue`; see below

Classes that can contain methods also have a new property called `$collectperformancedata`. This property is applied only when `$root.$prefs.$collectperformancedata` has the value `kCPDmarkedClasses`. If true, method execution performance data is collected for all methods in the class.

Assuming `$root.$prefs.$collectperformancedata` allows, the collected data is stored with each method in the class, with the following properties for each method:

\$callcount

The total number of calls to the method. You can only assign zero to `$callcount`, in which case Omnis also sets `$totalexecutiontime`, `$minexecutiontime` and `$maxexecutiontime` to zero

\$minexecutiontime

The execution time in milliseconds of the shortest call to the method

\$maxexecutiontime

The execution time in milliseconds of the longest call to the method

\$totalexecutiontime

The total execution time in milliseconds of all calls to the method

`$minexecutiontime`, `$maxexecutiontime` and `$totalexecutiontime` are floating point numbers.

Classes that contain methods, and can therefore collect method performance data, have two new methods:

`$clearperformancedata()`

Clears the performance data for all methods in the class

`$makeperformancedatalist()`

Returns a list containing the performance data collected for all methods in the class

There is a very small overhead when data is collected, while there is no impact on performance when performance data is not being collected.

Method Editor

Line Numbers

You can now display a line number for each method line in your code. You can enable line numbers using the View menu in the Method Editor.

Sequence Logging

Sequence logging allows you to record all method execution in Omnis or on the Omnis Server. If your Omnis Server is running multi-threaded mode, sequence logging can log method execution on multiple server threads.

Sys(3000) turns on sequence logging, and sys(3001) turns it off. Sequence logging writes every method command executed to a file in the Omnis directory. The name of the file is reported by an OK message when logging starts. Logging is thread-safe.

The log file can be up to 2mb in size, and when that limit is reached it discards all but the most recent 256kb and continues logging. Logged lines can be of any length. The log file name includes the date. The log file is UTF-8 and has a signature to mark it as such.

Each line in the file is prefixed with "N: " where N identifies the thread. Zero is the main thread. 1-N are Omnis Server threads.

When logging starts, Omnis writes a message to the log file, to allow the developer to identify logging being started and stopped.

Form Components

We have created two new components, the Accordion and FishEye, that take advantage of these new features and to further enhance your application development. The new components are available for window classes and remote forms.

Accordion Control

The Accordion control presents a list of hyperlink options, each of which has a header link that, when clicked, expands to show more information to the end user. In addition, each option in the list can include a further link option or an icon which users can click on. You can control how the options in the list expand and collapse, creating a very interactive selection tool for end users. The Accordion component is available for window classes (under External Components in the Component Store) and remote forms (FORMCORD component).

Populating the Accordion

The contents for the Accordion control is provided by a list variable specified in its \$dataname property. The list should have at least two columns to specify the Heading text and the expanded content or text, while you can add a third column for the additional Link text or icon. You can set the \$headingcolumn, \$contentcolumn, and \$headinglinkcolumn properties, to specify which list columns are used for the Heading text, Text content, and Link text, respectively.



For example, the following method defines a list with three columns to contain the Heading text, the content text, and the link text. You would typically add multiple lines in the list, which could be static data or loaded from a database.

```
; define iAccList (List) iAccHeading, iAccText, iAccLink (Chars)
Do iAccList.$define(iAccHeading,iAccText,iAccLink)
Do iAccList.$add("Heading 1",con("Some Text",chr(13),"over multiple
  lines ",style(kEscColor,kRed),"with word wrap",chr(13),"and some
  more lines",chr(13),"and some more"),"Action 1")
; and so on, to populate the Accordion list of options
```

The column specified in \$headinglinkcolumn can be a Character or Long Integer to either represent some hyperlink text or an icon id. In addition, when \$headinglinkcolumn specifies an icon id, you can include another column in the list to add a tooltip for each icon. The column containing the tooltips is specified in \$icontooltipcolumn.

For example, the following method is very similar to the previous example, but this time the link column is defined as a Long integer and icon IDs are added to the list contents.

```
; define iAccList2 (List), iAccHeading, iAccText (Chars), and
  iAccIconId (Long Int)
Do iAccList2.$define(iAccHeading,iAccText,iAccIconId)
For lNum from 1 to 100 step 1
Do iAccList2.$add("Heading A",con("Text ",lNum,chr(13),"over
  multiple lines ",style(kEscColor,kRed),"with word
  wrap"),1701+k16x16)
```

The icons can be added to the #ICONS system table in your library, but if you are using the control on a remote form, you must add the appropriate icon page name(s) to the \$iconpages property of the form.

Note in the example above, that your content column can contain styled text, assuming the \$styledtext property for the control is set to kTrue. The content column can also contain multiple lines of wrapping text, using a carriage return (chr(13)) as the line delimiter.

Expand mode and Animation

The \$expandmode property specifies what happens when the user expands or closes an entry in the Accordion control. The property is a constant as follows:

kACCexpandModeZeroOrOne	Zero entries or one entry must be open; when you open an option, any previous option will collapse automatically
kACCexpandModeOne	One entry must be open; the first option will open by default, then when you open another option, the previous option will collapse automatically
kACCexpandModeZeroOrMore	Zero or more entries can be open; options must be opened and closed manually
kACCexpandModeOneOrMore	One or more entries must be open; the first option will open by default, then further options can be opened and closed manually

If kTrue, the property \$fadetext forces the text to fade in or out when the entry is opened or closed, while the \$animationsteps property specifies the number of steps used to animate the opening or closing of an entry in the list (in the range 1-64).

Accordion Events

When the user clicks on the hyperlink (present if hyperlink column is present) the control generates evClick, with pLineNumber set to the clicked line number. Like many other list types, you could trap the line selected in the \$event() method for the control, and return the value of the heading text, the option text or some other value in another column in the list to make a selection in subsequent code.

FishEye Control

The FishEye control presents a row or column of icons to allow the end user to select an option by moving the mouse over the control and clicking on an icon. When the end user's mouse moves over the control, individual icons are enlarged and a text label is displayed for each icon. The FishEye component is available for window classes (under External Components in the Component Store) and remote forms (FormFish component).

Populating the FishEye

The contents for the FishEye control is provided by a list variable specified in its \$dataname property. The list should have at least two columns, one for the ID of the icons to be displayed in the control, the other column for the text labels for the icons. You can set the \$iconcolumn and \$textcolumn properties to specify which columns are used for the icons and text, respectively. \$iconcolumn is set to column 1 by default, so your text labels could be in column 2.

The icons can be added to the #ICONS system table in your library. To achieve the best magnification effect in the FishEye control, you should include the 48x48 version of each of the icons you wish to use. If you are using the control on a remote form, you must remember to add the appropriate icon page name(s) to the \$iconpages property of the form.

The following method defines the list for a FishEye control and adds some lines; you would normally add multiple lines, which could be static data or loaded from a database.

```
; define iFishList (List), iFishIcon (Long int), iFishText (Char)
Do iFishList.$define(iFishIcon,iFishText)
Do iFishList.$add(1704+k48x48,"Window")
Do iFishList.$add(1719+k48x48,"Remote Form")
Do iFishList.$add(1712+k48x48,"Object")
; and so on to build the list of options
```

Position and Expand direction

The \$position property determines how the icons and text in the control expand when the mouse hovers over the control, and together with the \$edgefloat property you can position the FishEye and set the direction in which it expands. \$position can be set to one of the kFishEyePos... constants, and \$edgefloat can be set to a kEFposn... constant. For example, with \$position set to kFishEyePosBottom, and \$edgefloat set to kEFposnLeftBottom, the FishEye control is positioned at the bottom of the form and the icons and text will expand upwards (as shown). You should experiment with the different \$position and \$edgefloat settings to achieve the effect and positioning you want.



Magnification and Stepping

The \$magnification property specifies the factor by which the FishEye control increases the size of the icon under the mouse; it should be set to a real number between 1.0 and 16.0. The greater the number, the greater the magnification.

The number of icons that expand to the left and to the right of the icon under the mouse is specified by \$steps; it should be an integer between 0 and 16. Therefore a low number, such as 0 or 1, will appear to make individual icons “popup” and you pass the mouse over the control, whereas a larger number will display a more gradual stepping effect. Again, you need to experiment with the \$magnification and \$steps properties.

The \$magnifyall property controls whether or not all the icons are expanded; it is set to kFalse by default which produces its characteristic gradual stepping effect as the mouse moves over the control. If true, the items are only displayed when the mouse is over the control, and they all have a fixed square size ($\$magnification * \$height$ for horizontal positions or $\$magnification * \$width$ for vertical positions) and \$steps is ignored.

Text Labels

You can specify the color of the text in \$textforecolor. You can set the color and pattern for the background of the labels in \$textbackcolor and \$textpattern. In addition, you can specify the transparency of the label background by setting the \$textalpha property; this is a numeric value in the range 0-255, where 0 is completely transparent and 255 is opaque.

FishEye Events

Clicking on an icon in the FishEye control generates an `evClick` event with `pLineNumber` set to the line number of the clicked entry. You could return the value of the label or some other value in another column in the list based on line selected.

Accessing the Windows Registry

The new Omnis RegAdmin component is a non-visual object that gives you access to the Windows Registry allowing you to manage system information on the client machine. As the Windows Registry is applicable to the Windows platform only, all the functions return an error message on Mac OS X and Linux.

WARNING: The registry contains system information that is vital to your computer, and an incorrect change to your computer's registry could render your computer inoperable. We strongly recommend that you back up the registry before making any changes to it, especially if you are creating and testing code that will modify your registry or your customers' registry settings. Alternatively, you can create a restore point using System Restore which will allow you to undo changes to your system settings if required. For further information about the registry, backing up and creating restore points, refer to the Windows Help.

Note that under Windows Vista access to files in the Windows Registry is limited, unless you turn off User Account Control (UAC), which may not be possible or permissible on end user PCs.

Omnis RegAdmin

The new Omnis RegAdmin component is a non-visual object that allows the end-user to manage Keys and Values within the Windows Registry. The Windows Registry provides a means for storing settings and options for the Windows Operating System and its applications. It consists of a number of what are termed *Keys* and *Values*. Keys are similar to the notion of folders and are used to structure the registry. Each Key can contain any number of sub-keys and values. Values consist of a name and data pair and are the means for storing settings. The Windows Registry structure is split into a number of logical sections including some default Keys, often termed as "hives". These all are pre-fixed by 'HKEY', for example, HKEY_LOCAL_MACHINE, or HKEY_CURRENT_USER.

Creating a RegAdmin Object

To access the functions within RegAdmin, you will need to define a RegAdmin object. You can do this using any of the following methods:

- Declare an Omnis Object variable in the Method Editor and set the subtype to the RegAdmin object.
- Declare an Object variable in the Method Editor and initialize it to point to a RegAdmin object via notation.
- Define an Object Reference variable in the Method Editor and initialize it to a RegAdmin object via notation and the `$newref()` function.

Opening a Key

Once you have an instantiated object or object reference the next step is to point your object to a key in the registry. This is done using the `$openkey()` method. This method takes two parameters: the first can be either a `kRegAdminKey` constant (see below), or an existing `RegAdmin` object. The second parameter is optional and specifies the path to the key you want to open. For example, if you provide the constant `kRegAdminKeyClassesRoot` as the first parameter and “Applications/omnis.exe” as the second, you would open up the key `HKEY_CLASSES_ROOT\Applications\omnis.exe`. The method returns the flag indicating if the specified registry key was opened successfully, 1 for success.

The following Omnis code creates an Object reference to the `RegAdmin` object and assigns it to `iRegAdminRef` Object Reference variable. The `$openkey()` method sets the Object Reference to point to a key in the Windows Registry. If this call fails the OK message reports the error. When you’ve finished using your `RegAdmin` object, you have to call the `$closekey()` method to free up the associated memory.

```
Do $extobjects.RegAdmin.$objects.RegAdmin.$newref() Returns
    iRegAdminRef
Do iRegAdminRef.$openkey(
    kRegAdminKeyClassesRoot,'Applications\omnis.exe') Returns #F
If flag false
    OK message Failed to open ... {[iRegAdminRef.$getlasterror]}
End If
```

Navigating the Registry

You can open a registry key providing a Root Key handle (`kRegAdminKey...` constant) or by providing an existing open key. The only caveat is that you can’t pass in the calling object as the first parameter to the object.

Creating new Keys and Values

To create a new key you must execute the `$createkey()` method. It takes one parameter and a second optional parameter. The first parameter can be either an existing `RegAdmin` object, or a root key handle constant (`kRegAdminKey...`). The second optional parameter can specify a path to a subkey, separating keys via the ‘\’ separator. This call will create and open a new key if not present in the registry. In the event that the key specified by the first and second parameters does exist, that key will be opened.

You can add additional values to keys using the `$setvalue()` method, which takes two parameters and accepts an additional optional third parameter. The first parameter specifies the name of the value you wish to create. The second takes a `kRegAdminValue...` constant and specifies the type of value you wish to add to the key. The third and optional parameter lets you set an initial value. The format of this value will be dependent upon the type of value you’re adding to the registry and is as follows:

Binary Value	Omnis Binary
Multi String Value	Omnis List of Omnis Characters
String Value	Omnis Character
Expandable String	Omnis Character
DWORD Value	Omnis Long Integer

Retrieving Key and Value names

Once you have opened a key successfully, you can use the \$listskeynames() method to query the register and return a list of subkeys within your existing key. This method takes no arguments and returns a list of Key Names in the opened key as an Omnis list of characters.

To retrieve a list of Value names, the \$listvaluenames() method can be used. This method is similar to \$listskeynames() except that it returns a list of Value names rather than Key names.

Reading Values

You can read values using the \$getvalue() method, but the return value is dependent on the type of registry value you are retrieving. Therefore, to determine the type of the registry value, you can execute the \$getvaluetype() method passing in the name of the parameter as the first argument. This returns a kRegAdminValue... constant specifying the type of the registry value.

Once you have returned the type of the value in the registry, you can call the \$getvalue() method with a suitable variable return, as follows:

```
Do iRegAdminObj.$getvaluetype(iValueName) Returns iValueType
Switch iValueType
  Case kRegAdminValueMultiSz
    Do iRegAdminObj.$getvalue(iValueName) Returns lList
  Case kRegAdminValueBinary
    Do iRegAdminObj.$getvalue(iValueName) Returns lBinary
  Case kRegAdminValueNone
    Calculate lCharacter as ''
  Default
    Do iRegAdminObj.$getvalue(iValueName) Returns lCharacter
End Switch
```

Deleting Keys and Values

To delete a key within the registry you can execute the \$deletekey() method. This takes a single parameter, which is the name of the key you want to delete. This either has the name of a sub-key or the path to the key relative to the currently open key. For example, if you want to delete the HKEY_CLASSES_ROOT\Applications\omnis.exe key, you can either

open HKEY_CLASSES_ROOT and execute \$deletekey('Applications\omnis.exe') or open HKEY_CLASSES_ROOT\Applications and call \$deletekey('omnis.exe').

If the key you're attempting to delete contains any number of sub-keys, an error is returned and the key is not deleted.

WARNING: When you delete a Windows registry key, the key itself and its values are permanently deleted.

To delete a value from a key, you can use the \$deletevalue() method. This takes a single parameter, which specifies either a value name or the path to the value name. The path is relative to the current open key.

WARNING: Deleting a key value is permanent.

Error Handling

Most RegAdmin object methods return a flag to indicate whether or not the call was successful. In the event that the flag returned is false there are two error handling method calls which you can use to report the error. The \$getlasterrorcode() method returns the Windows error code associated with the error, and \$getlasterror() returns a description of the error provided by the Windows operating system.

DAMs

OpenBase

This section contains the information you need to access a database using the OpenBase object DAM including server-specific programming, data type mapping and troubleshooting. For general information about logging on and managing your database using the SQL Browser, refer to the *Omnis Programming* manual.

Properties and Methods

Session Methods

Methods	Description
\$newid()	<p>SessionObj.\$newid(pTable,pColumn)</p> <p>This method retrieves a unique key from the database for the specified table and column. It returns a numeric value, floating decimal point corresponding to a 64 bit integer. Using \$newid is the equivalent to creating a statement object and executing a statement like: "newid for ". Calling \$newid() may destroy active cursors (result sets) in statement objects, but should not otherwise interfere with statement objects. Example:</p> <pre>IUniqueID = tOpenBaseSession.\$newid('ANIMAL', 'ANIMAL_ID')</pre>

Session Properties

Property	Description
\$obhost	This read-only character property contains the DNS host name of the server to which the session is connected.
\$obdatabase	This read-only character property contains the name of the database to which the session is connected.
\$obversion	This read-only character property contains the version of the OpenBase database to which this session is connected.
\$obpassivetransaction	This read-write boolean property contains the setting for passive transactions. The default value for new sessions is true. Passive transactions generally help you avoid lockups when multiple clients access the same table within the context of open transactions.
\$obtimezoneoffset	This read-only character property contains the timezone offset for the client, suitable for appending to date time literals before inserting into the database. For Pacific Standard Time, this property would contain “-0800” (no quotes, note leading space).

Statement Properties

Property	Description
\$obpreparedstatement	This read-only character property contains the last statement prepared. This may be different from what your library prepared if the statement has been adjusted by the DAM to be syntactically more correct.
\$obexecutedstatement	This read-only character property contains the last statement actually sent to the OpenBase server. All variables are bound on the client-side, so this will have bind variables substituted.
\$obfilterrowid	This read-write boolean property determines whether the statement object should filter out the <code>_rowid</code> column when executing <code>select * SQL</code> statements. The default value for new sessions is true. If you use the <code>_rowid</code> column as your primary key (as many OpenBase developers do) and you perform <code>select * queries</code> , you may want to set this property to false for each new statement object.

Logging on to OpenBase

Please note that when connecting to an OpenBase server, the required database name is specified as part of the hostname, e.g.

```
Do sessObj.$logon('dbname@192.168.0.10','myuser','mypwd','sess1')
```

The host name parameter also allows an optional software ID and client name to be specified. The software ID is used by OpenBase to restrict access by applications based on

user. The client name shows up in the Process Viewer window of OpenBase Manager and can help sort out actual client machines with connections. The valid formats for the host name are:

@;;	e.g. TestDB@example.com ;My Application;Joe's Computer e.g. TestDB@example.com ;My Application
@;;< client name >	e.g. TestDB@example.com ;;Joe's Computer
@	e.g. TestDB@example.com

OpenBase Data Type Mapping

When connecting to OpenBase using the OpenBase DAM, it is helpful to understand how Omnis Studio types are mapped to OpenBase types and vice versa. Note that the mapping is not one-to-one, as there is not a perfect correspondence among types in both systems. When copying tables between OpenBase sessions using the SQL Browser, you may have to edit column types in the destination table to get exact copies. The OpenBase server is flexible in this respect, and will automatically do things like converting floating point values to longlong if needed. So it is possible to get exact copies of tables with the SQL Browser by giving it a few hints.

OpenBase to Omnis

The following table describes how OpenBase column types are mapped to Omnis Studio variable types by the OpenBase DAM.

OpenBase Data Type	Omnis Data Type
char (varchar)	Character
float (double)	Number - floating dp
int	Number - long integer
long	Number - long integer
long long	Number - 0 dp
money	Number - 2 dp
date	Date time - Short date
time	Date time - Short time
datetime	Date time - Date time
object	Binary
boolean	Boolean
binary	Character

The OpenBase binary type, new to version 7, is primarily intended for WebObjects keys and is probably not very useful to Omnis developers.

Omnis to OpenBase

The following table describes how Omnis Studio variables are mapped to OpenBase columns by the OpenBase DAM. This mapping occurs primarily when tables are created in OpenBase or after repeated calls to the \$coltext() session method.

Omnis Data Type	OpenBase Data Type
character(n), n <= 4096	char
character(n), n > 4096	object
Boolean	boolean
Short Date	date
Short Time	time
Date Time (all subtypes)	datetime
Sequence	integer
Integer	integer
Number	float
Picture	object
Binary	object
List	object

The OpenBase DAM will quote the literals generated by numeric (floating point) bind variables, making it safe to insert such variables into char, int, long, longlong, and float columns in your OpenBase database.

Date & Time Types

This section describes the date, time, and datetime OpenBase column types and the implications these types have for developers working with the OpenBase DAMs. Of particular interest is how clients and servers in different time zones work together. Commented sample code shows how various features work.

OpenBase Date Columns

An OpenBase date column contains a calendar date, e.g. November 3, 1970. A date is reported the same regardless of the time zone settings of the server or client. If you insert a date value from Omnis using bind variables or by rolling your own SQL statement, the same date will be stored on the server and the same date will be returned when selected later.

OpenBase Time Columns

An OpenBase time column contains a wall-clock time, e.g. "15:17". A time is reported the same regardless of the time zone settings of the server or client. If you insert a time value from Omnis using bind variables or by rolling your own SQL statement, the same time will be stored on the server and the same time will be returned when selected later.

OpenBase Datetime Columns

An OpenBase datetime column contains a UTC timestamp, e.g. "1988-07-15 17:04:00 +0000". Inside the server, the timestamps are normalized to GMT. A timestamp is,

however, reported by client software in the client's local timezone. So a client in California (PST) reading the preceding datetime example would display "1988-07-15 09:04:00 – 0800". Semantically, this makes sense because a timestamp defines a specific instance in time that spans time zones. A timestamp does not just define a calendar date and a wall-clock time. How that instance in time is referenced depends on the time zone in reference is made.

Date time values in Omnis do not contain timezone offset information. Care needs to be taken when inserting and retrieving these values to ensure that you get the results you want. When the DAM reports a datetime value from OpenBase, it adjusts the value to reflect the client's time zone. So, an Omnis client in California will report the preceding datetime example as 1988-07-15 09:04:00. Notice that the time zone offset is dropped because a datetime value in Omnis does not contain that information.

When "date time" values are inserted through the DAM using bind variables, the DAM will append the appropriate time zone offset to the OpenBase datetime literal.

However, if you roll your own SQL statements, you are responsible for appending the appropriate timezone offsets. Otherwise, the server will interpret the datetime literal as GMT. Fortunately, the OpenBase v3 DAM can help you append the client's timezone information when you create a datetime literal. An offset string for the client's timezone is provided by the session object's \$obtimezoneoffset read-only property:

```
Calculate lDatetimeLiteral as con(''1988-07-15 09:04:00',
    tobSession.$obtimezoneoffset, '')
```

Note: the single quotes that surround the datetime literal.

One other issue to note is that OpenBase datetime values do not support hundredths of a second. The DAM drops fractional second information when converting bind variables. Your code should ignore the fractional second portion of date time values when creating datetime literals for OpenBase.

Table and Column Names

Identifiers are used for table names and column names, among other things. Identifiers in OpenBase can be any combination of letters, numbers, and the underscore character. OpenBase identifiers are currently case sensitive, so "MyTable" (no quotes) is different from "MYTABLE" (no quotes). This may change in a future version of OpenBase.

You may also use SQL keywords as OpenBase identifiers. For example, "name" (no quotes) is a perfectly legitimate column name in OpenBase.

Quotes

Some database systems allow you to use SQL keywords by enclosing them in double quotes, e.g. "name" (with quotes). OpenBase does not currently recognise quotes around identifiers. The Omnis Studio SQL Browser does not currently generate SQL statements that include quoted identifiers, so there are no problems using it with OpenBase.

Be cautious with Omnis Studio Schema objects. They allow you to use quoted identifiers as server table and server column names. OpenBase is not compatible with this feature. When you drag a schema that includes quoted identifiers to an OpenBase session in the SQL

Browser, an error will be reported about an identifier being expected. Remove the quotes from the table and column names in the schema and the problem should be solved.

OpenBase Transactions

There are two important things to understand about OpenBase transactions:

If you want to have multiple SQL statements in a single transaction, you must explicitly start the transaction. If you haven't started a transaction, the server commits each statement as it receives and executes it.

OpenBase has an option for transactions to be passive. Normally, transactions do not allow dirty reads. When one connection performs an update, the change is visible within the transaction immediately. If another connection performs an update on the same table, the server will block until there is a COMMIT or ROLLBACK of the transaction in the first connection. This is to ensure serialisation and avoid unnecessary deadlocks.

A passive transaction, by contrast, does not obtain locks while a transaction is in process and makes no changes visible to any connection until a COMMIT is issued.

Passive transactions allow dirty reads until the COMMIT occurs, and do not block. Yet, passive transactions are still guaranteed to be atomically correct and serialised correctly if they succeed.

Implications for Omnis Studio

The OpenBase object DAM supports three transaction modes: *server*, *automatic*, and *manual*. In both server and automatic modes, OpenBase will commit each statement as it receives and executes it. In manual mode, OpenBase will commit each statement as it receives and executes it unless you have begun a transaction with \$begin() and have not concluded it with either \$rollback() or \$commit().

The OpenBase DAM defaults to using passive transactions on each new connection. This allows your clients to update the same tables without locking up and without any need for an investigation of transaction semantics. You can change this by setting the \$obpassivetransactions property of the OpenBaseSESS object to kFalse.

Troubleshooting

The OpenBase DAM's type mapping scheme depends on some small fixes in OpenBase SQL 7.0.5. Version 1.1 and later of the OpenBase DAMs also depend on OpenBase SQL 7.0.7. As such, the OpenBase DAMs should be used with OpenBase SQL 7.0.7 or later.

Miscellaneous Enhancements

External Component Interface

The Omnis External Component interface now supports animation and transparency allowing developers to create new interactive and visually rich components. We have created two new components, the Accordion and Fisheye, which take advantage of these new features in the xcomp/webcomp API. The new components are available for window classes and remote forms and described earlier in this manual.

Form Component Enhancements

The following enhancements apply to either Remote form or Window class components or both.

Remote Form Controls

Remote form fields and other web components have been significantly enhanced and now comply with the expected appearance and behavior across all supported platforms, including Windows Vista and Mac OS X Leopard.

There were many small discrepancies with how various remote form fields and other components looked and behaved on different platforms, especially when compared to the equivalent window class fields. Many of these small discrepancies have been rectified in this release, and now all the web components conform to the expected appearance and behavior for each platform supported in Omnis. For example, on Mac OS X Leopard all types of list now display alternate colored lines, and on Vista headers in headed lists and pushbuttons are highlighted when you move your mouse over them. Overall, this means end users will have a more standardized experience when using an Omnis application.

Remote Form Field Borders

The \$bordercolor property has been added to remote form fields, which means you can now change the color of borders on entry fields, masked fields, and so on.

UserInfo

There is a new method called \$getuserdata(cUserKey) that lets you return the user data with the specified key.

QuickTime

There is a new property in the QuickTime movie player called \$centermovie which allows you to center a movie in the movie player field area.

Gradients

There are four new patterns that support drawing gradients (left to right, right to left, top to bottom and bottom to top), and which appear at the bottom of the pattern palette where they apply, such as window backgrounds, and rectangle background objects. The gradients use the foreground and background colors as the start and end colors of the gradient.

Copying Text in Fields

The standard edit field, multi-line edit field and combo box, for both fat client and the web client, can allow the end user to copy text from the respective field. There is a new property called `$allowcopy` which can be enabled for all standard field types to allow data to be copied, even if the field is disabled.

Encryption Functions

`encxtea()`

`encxtea(binary, key)` returns the binary result of encrypting *binary* using the eXtended Tiny Encryption Algorithm (XTEA) with the binary *key*; the key must be 128 bits long.

`decxtea()`

`decxtea(binary, key)` returns the binary result of decrypting *binary* (previously encoded using `encxtea()`) with the binary *key*; the key must be 128 bits long.

Apple Events

`$runapplescript()`

A third optional parameter has been added to the `$runapplescript` method, which returns any error string from running the compiled script.

Lists

Dynamic Column Headers

End users can now reorder the columns in headed lists by clicking and dragging the column headers, as appropriate. This enhancement applies to headed lists in the thick and thin client.

The headed list field has a new runtime property called `$displayorder` (or `$::displayorder` in the thin client object) which is a comma-separated list of column numbers, indicating the order in which columns are displayed by the headed list. Initially, `$displayorder` is set to 1,2,..., up to and including the value of `$colcount`, that is, the number of columns in the list. This is reset to the initial value whenever you change the number of columns. You can set this property using the notation, to reorder the columns.

In addition, headed lists have a new property called `$candragdisplayorder` (or `$::candragdisplayorder` in the thin client object). If true, and `$enableheader` is also true, the user can drag and drop a column in the header to reorder the columns in the headed list.

When the user changes the order of the column headers, the column order stored in `$displayorder` is changed accordingly. If you allow the user to drag and drop in the column heading, you can then use `$displayorder` to save and restore the order the user has selected.

Display Order Events

In addition, there is a new event, `evHeadedListDisplayOrderChanged`, with an event parameter `pDisplayOrder` (containing the new `$displayorder`), which the headed list box receives when the display order has been changed using drag and drop.

Even after changing `$displayorder`, column numbers in all properties and events related to the headed list box are the original column number specified in the library in design mode. This means that changing `$displayorder` does not require changes to the methods that manipulate the headed list box.

Displaying Totals in Headed Lists (Web only)

Headed lists in the Omnis Web Client can now have footer buttons to display totals data about the columns. Note this feature is available for headed lists for Remote forms only and not the equivalent component for Window classes.

There are a number of new properties in the headed list web component to allow you to display totals.

- \$hidefooter**
defaults to `kTrue` for compatability but set to `kFalse` to see the footer buttons.
- \$footerselected**
set this to `kTrue` if you want to only include the selected lines in the total
- \$setfootercalc** (runtime only)
takes a comma separated string with a max of 3 values – the column number, a report constant (`kTmAvergae`, `kTmCount`, `kTmMaximum`, `kTmMinimum`, `kTmTotal`), followed by an optional string which is displayed in the footer before the total
E.g. Do `$cinst.$objs.headedListObj.$setfootercalc.$assign('2,kTmTotal,Total')`
- \$enablefooter**
set this to `kTrue` to generate the event `evHeadedListFooterBtnClick` in Omnis and return the column number and total

Searching Lists

The behavior for searching list fields using the keyboard has changed in this version. There is a new Omnis root preference called `$oldlistsearching` to control the list search behavior.

In previous versions of Omnis, you were able to tab into or click on a list and search the first column of the list by typing a few characters. The focus in the list would jump to the line containing the characters you typed. In this case, Omnis stored the characters you typed into a search buffer, regardless of the delay between each character you typed, and tried to find a match in the list. You could continue to type extra characters and Omnis would add these to the search buffer. In addition, you could use the `+` and `-` keys to find the next and previous matches in the list, and you could use `*` to represent wildcards in your search.

The old list searching behavior is enabled only when `$oldlistsearching` is set to `kTrue`. By default, `$oldlistsearching` is set to `kFalse` meaning that the old list searching behavior no longer works in the various list fields, including list boxes and headed list fields. You can still search in a list field by typing a few characters, but if there is a delay in your typing the search buffer is reset and you are able to type another string to search the list again. The `+`, `-` and `*` keys (plus, minus, and asterisk) are now treated as normal search characters, rather than having a special function.

List Column Totals

The `$totc()` list method has a new parameter to allow you to total the selected lines only. `$totc(expression[,bSelectedOnly])` returns the total of the expression for selected lines only if `bSelectedOnly` is `kTrue`; the default is `kFalse` when all lines are totaled.

Droplists

You can now use styled text in droplists on remote forms, which means you can add icons and text formatting to lines in droplists.

Toolbars

Combo boxes

Combo boxes can now be used in unified toolbars on Mac OS X. They have the rounded edit field appearance, like the finder toolbar search box.

The `$iconid` property is now available, and applies to OSX combo boxes in toolbars only. It specifies the icon drawn to the left of the control, where a click opens the menu containing the combo box list.

- When set to zero, the icon is the standard OS X search icon
- When set to a valid value, the 16x16 icon selected replaces the search icon
- When set to an invalid value, the control just shows a down arrow like a combo box

To use a combo box for searching (for example), check for `evAfter` with the next event of `evOK`; this is generated when you hit return with the focus in the combo box.

On *all platforms*, combo boxes now have a content tip when there is no text label drawn in the toolbar; the content tip uses the value of the text property.

Toolbar spacers

Toolbar separators have a new property `$flexible` that can be set to `kTrue` when `$blank` is `kTrue`:

- \$flexible**
If true, and `$blank` is also true (that is, no dividing line is shown), and the toolbar is on a window, the separator expands in width to use docking area space not occupied by other toolbar objects. If more than one separator is flexible, the unused space is shared equally between them.

A typical use of `$flexible` is to place a single separator in the toolbar, and set `$blank` and `$flexible` to `kTrue`. The result is that any controls to the right of the separator become right justified.

`$flexible` applies on all platforms.

Omnis Data Bridge

Omnis SQL and DML connections

Connections to an Omnis datafile can now be made concurrently using both Omnis SQL and the Omnis Data Bridge using a single serial number quota. Omnis no longer treats host names and IP addresses from the same machine as different entities.

Omnis Datafile List

There is a new Omnis Root method, called `$getodbfilelist(lList,[cHostIP])`, that returns a list of Omnis datafiles registered with the Omnis Data Bridge. `cHostIP` should be of the form `'odb://ip-address:port'`, or if omitted the value in the `$odbserver` root preference is used.

FileOps

`$filelist()`

The FileOps function `$filelist()`, which returns a list of files, directories, and volumes, has an additional parameter to return any error message from the current system.

The syntax is now:

```
FileOps.$filelist(iInclude,cPath[,iInformation,cFilter,&cErrorText])
```

Errors are reported to `cErrorText`.

oXML

New Format constants

The format parameter for the `$savefile()` and `$savebinary()` methods in the oXML object was a Boolean, but is now an integer and can take one of a number of new constants.

The new format constants are:

- kXMLFormatNone**
The output XML is not formatted (that is, no tabs and carriage-return linefeed sequences are inserted)
- kXMLFormatBasic**
The output XML is formatted by the insertion of tabs and carriage-return linefeed sequences
- kXMLFormatFull**
The output XML is formatted by the insertion of tabs and carriage-return linefeed sequences; in addition, text nodes are formatted by removing all leading and trailing spaces, as well as tabs, carriage returns and linefeeds

Omnis Application Window

Omnis Background Image

There is a new Omnis root preference, `$backgroundiconid`, that allows you to add an image to the background of the Omnis application window.

You can use an image from the Omnispic icon datafile, but if you wish to use your own image you must add the image to the Userpic icon datafile or your own data file located in the icons folder. We recommend that you do not add your own icons or images to the Omnispic datafile.

You can add an image to the Userpic file or your own data file by creating a new 'Image Page' (rather than an icon page). An image page stores a single image and can be any size you like, such as 1280 x 1024, into which you can paste a suitable background image. The image will be displayed centered on the main Omnis application window.

When you have selected or added an image, you need to set the `$backgroundid` property to the ID of the image page. You cannot use an icon in a #ICONS table.

You can download some "wallpaper" images from the Omnis website which can be displayed on the background of your Omnis application window. Alternatively, you may like to add an image with your own company logo, or your image could be a single solid color. This technique would allow you to add further branding to your own applications.

Reports

Report Toolbar preferences

There are two new Omnis root preferences in the `$root.$prefs` devices group, called `$reporttoolbar` and `$reporttoolbarpagepreview`, that allow you to select the buttons on the screen report and page preview toolbar for user reports.

Disable Report Copy

There is a new Omnis root preference in the `$root.$prefs` devices group, called `$disablereportcopy`, that allows you to disable the copy via selection feature of screen reports and page previews for user reports.

Omnis Help

Help Folder

You can open the Omnis Help window from within your application using the `$exechehelp()` method, which is a method under `$root`.

The method now has an additional parameter:

```
$exechehelp(cInstName, cWindowTitle, cHelpFolder  
            [, cDocumentName, cTopic, bIDEhelpFolder=kFalse])
```

When `bIDEhelpFolder` is true the Omnis help is stored in the 'idehelp' folder rather than 'help' within the Omnis folder. The idehelp folder was introduced when Vista support was added so that the Omnis help files could be installed in the read-only Program Files directory.

Docking Areas

Toolbar Docking Area Height

There is a new library preference called `$windowsizeexcludesdockingarea` that allows you to ignore the height of the toolbar docking area when specifying the position of a window.

If true, the width and height of a window exclude the relevant dimension of the toolbar docking area (this does not affect windows under Mac OS X with a standard OS X top toolbar since they are excluded from the docking area automatically).

The new preference defaults to true in new libraries, but is false in existing libraries for compatibility.

Mac OS X

Startup Parameters

On Mac OS X, you can now run Omnis from the command line passing in startup parameters to the Omnis program. The first parameter is interpreted as a library name which Omnis tries to open on startup. The path has to be in the POSIX format.

To run Omnis from the command line on Mac OS X, you have to use the Terminal application, found in `/Applications/Utilities/`. You need to call Omnis from inside the Omnis package. For example, if your package is called Omnis Studio 5.0, you could navigate to or call directly:

```
/Applications/Omnis Studio 5.0.app/MacOS/Contents/
```

You can then run Omnis from the command line by typing:

```
./Omnis
```

You can use an absolute path to the Omnis program, for example, by typing:

```
/Applications/Omnis\ Studio\ 5.0.app/MacOS/Contents/Omnis
```

which will launch Omnis. Note that in Terminal, spaces in path names have to be escaped with backslash, for example, `Omnis\ Studio\ 5.0.app`.

In addition to the command to start Omnis from the Terminal application, you can pass in a number of arguments or parameters, which are read by Omnis on startup. Multiple arguments must be separated by a space.

The first argument is interpreted as a library name, and Omnis will attempt to open the library. If the library does not exist or cannot be found, Omnis displays an error message. The library name must include the extension associated with the library, which in most cases will be `.lbs` in the context of cross-platform applications. The path to the library has to be specified in POSIX form and it has to include the full path to the library. If you don't want the first argument to be interpreted as a library name, you must prefix the argument with a hyphen (-). The startup parameters can be retrieved by calling the `sys(202)` function.

Examples

Calling the following from within the `/Applications/Omnis Studio 5.0.app/MacOS/Contents/` directory:

```
./Omnis /Applications/Omnis\ Studio \  
5.0.app/MacOS/Contents/welcome/button.lbs
```

will launch Omnis and load the buttons.lbs example library. Calling sys(202) will return:
/Applications/Omnis Studio 5.0.app/MacOS/Contents/welcome/button.lbs

Entering the following command will launch Omnis without any attempt to open a library.
In this case, sys(202) would return -arg1 arg2 arg3.

```
/Applications/Omnis\ Studio \ 5.0.app/MacOS/Contents/Omnis -arg1  
arg2 arg3
```

Typing the following:

```
./Omnis MacOS:Applications:Omnis Studio  
5.0.app:Contents:MacOS:welcome:button.lbs
```

Omnis would launch but an error message would result.

ODBC Driver for Omnis Data Files

DSN-less Connectivity

You can now use the Omnis ODBC driver to connect to an Omnis 7 or Studio datafile without the requirement for a System or User Data Source Name.

To use the Omnis ODBC driver without a System or User Data Source Name, it is necessary to specify the connection information using an ODBC connection string. A DSN-less connection will only be possible where the third-party application allows a connection string to be supplied in place of a user or system DSN.

The Omnis ODBC Driver supports the following keywords:

Driver	String value. Required
DataFilePath	String value. Required
Description	String value. Not required
Username	String value. Required
Password	String value. Required
TruncateStrings	1 or 0. Optional
NoCatalogues	1 or 0. Optional

An example of an ODBC connection string passed to the Omnis ODBC driver might be:

```
"Driver=Omnis ODBC Driver; DataFilePath=c:\TRAVEL.DF1;  
Username=myuser; Password=myspassword"
```

The password can be supplied in encrypted form if required, that is, it can be copied from the Windows registry or the odbc.ini file after creating a User or System DSN.

The optional keywords have the following effect:

* **TruncateStrings**

When set to 1, this allows the driver to work correctly with the Microsoft Data Transformation Services (DTS) application which currently has a problem retrieving

character strings larger than 8000 characters. Character strings will be truncated at 8000 bytes.

You should use this if you encounter an error similar to: “HY090: Invalid string or buffer length”.

* **NoCatalogues**

When set to 1, this enables a patch which strips a catalogue-qualified table name supplied in calls to SQLTables and re-substitutes it into the result set returned to Crystal Reports.

You should use this with Crystal Reports 10 and later where the connection does not display a list of tables.

Secure Web Communications

Support for secure connections using SSL has been added to the existing package of web commands. In addition, there is a new set of commands that allow you to communicate with an IMAP email server.

The HTTP, FTP, SMTP and POP3 client commands that establish a connection to a server all have two new arguments, which control if and how a secure connection is used.

The new and updated Web Commands are summarized here and described in more detail in the Omnis Help (press F1).

SSL Security

The external commands in Omnis Studio that allow low-level web communications have been enhanced to allow support for secure Internet-based connections using Secure Sockets Layer (SSL) technology. The FTP, HTTP, SMTP, POP3, and IMAP client commands that establish a connection to a server now allow you to control if and how a secure connection is used. The commands which allow secure connections are:

FTPConnect	IMAPConnect (new)
HTTPGet	POP3Connect
HTTPOpen	POP3Recv
HTTPPost	POP3Stat
HTTPSetProxyServer	SMTPSend

Two new parameters, called *Secure* and *Verify*, have been added to these commands to support secure connections. The new parameters behave as follows:

Secure

is an optional Boolean parameter which indicates if a secure connection is required to the server. Pass `kFalse` for non-secure (the default). Pass `kTrue` (value 1) for a secure connection; this enables the *Verify* option. In addition, you can pass value 2 to some of

the commands to enable specific types of authentication. To use a secure connection, OpenSSL must be installed on the client system: see below.

❑ *Verify*

is an optional Boolean parameter which is only significant when Secure is not kFalse. When Verify is kTrue, the command instructs OpenSSL to verify the server's identity using its certificate; if the verification fails, the connection will not be established. You can pass Verify as kFalse, to turn off the verification; in this case, the connection will still be encrypted, but there is a chance the server is an impostor.

See the Omnis Help (press F1) for full details about each updated web command and how to enable secure connections.

OpenSSL

Support for secure connections is enabled using the open source library called OpenSSL, which must be available on the client system. This is installed and enabled on Mac OS X and most Linux distributions by default. For Windows however, you or your end-users need to download and install the OpenSSL binaries, which are available from the OpenSSL organization at: <http://www.openssl.org/related/binaries.html>

Note: The OpenSSL toolkit is licensed under an Apache-style licence, which means that you are free to use it for commercial and non-commercial purposes, but you must comply with certain license or legal conditions. See the OpenSSL website for further details.

Certificate Authority Certificates

In order to perform the verification (when the Verify parameter is kTrue), OpenSSL uses the Certificate Authority Certificates in the cacerts sub-folder of the secure folder in the Omnis folder. If you use your own Certificate Authority to self-sign certificates, you can place its certificate in the cacerts folder, and OpenSSL will use it after you restart Omnis.

IMAP

There are a number of new external commands to support communications with IMAP mail servers. The new IMAP commands can use a secure connection. The new commands are prefixed with the letters 'IMAP' and are summarized below, but are described in more detail in the Omnis Help (press F1).

Command	Description
IMAPCheck	Sends a CHECK command to the IMAP server which requests a checkpoint of the currently selected mailbox. You must select a mailbox using IMAPSelectMailbox before using this command
IMAPConnect	Establishes a connection with an IMAP server, which must support IMAP4rev1; requires a server name, username, and password; if successful, it returns the socket opened which can be used with the other IMAP commands; you must call IMAPDisconnect when finished with the IMAP server
IMAPCopyMessage	Copies a message from the currently selected mailbox to another mailbox, using the UID COPY command. You must select a mailbox using IMAPSelectMailbox before using this command
IMAPCreateMailbox	Creates a new mailbox on the IMAP server; requires a socket number created using IMAPConnect and a mailbox name
IMAPDeleteMailbox	Deletes a mailbox and the messages it contains; requires a socket number and a mailbox name
IMAPDisconnect	Closes a connection to an IMAP server; requires a socket number
IMAPExpungeMessages	Permanently removes all messages that have the \Deleted flag set from the currently selected mailbox; requires a socket number
IMAPListMailboxes	Returns a list of a subset of mailbox names from the complete set of all names available to the client by sending a LIST command to the IMAP server; requires <i>socket, refname, mailboxname, list</i> ; the list must be predefined containing 7 columns
IMAPListMessages	Gets a list of messages in mailbox previously selected using IMAPSelectMailbox; requires <i>socket, list</i> ; the list must be predefined containing 9 columns
IMAPListSubscribedMailboxes	Returns a list of a subset of mailbox names from the

Command	Description
	complete set of all subscribed names available to the client by sending an LSUB command to the IMAP server; requires <i>socket</i> , <i>refname</i> , <i>mailboxname</i> , <i>list</i> ; the list must be predefined containing 7 columns
IMAPNoOp	Sends a NOOP command to the IMAP server; you can use the command to poll the server to get status updates via untagged responses returned to the <i>responselist</i> parameter if present
IMAPRecvHeaders	Receives the headers for a specified message in the currently selected mailbox; you can pass the received headers to the MailSplit command to parse them; requires <i>socket</i> , <i>messageuid</i> , <i>headers</i>
IMAPRecvMessage	Receives a specified message in the currently selected mailbox; you can pass the received message to the MailSplit command to parse it; requires <i>socket</i> , <i>messageuid</i> , <i>headers</i>
IMAPRenameMailbox	Renames a mailbox; requires <i>socket</i> , <i>oldmailboxname</i> , <i>newmailboxname</i>
IMAPSelectMailbox	Makes a mailbox the currently selected mailbox, allowing other IMAP commands to operate; requires <i>socket</i> , <i>mailboxname</i> , <i>messages</i> , <i>recent</i> , <i>uidnext</i> , <i>uidvalidity</i> , <i>unseen</i>
IMAPSetMessageFlags	Adds or removes flags for a message in the currently selected mailbox; requires <i>socket</i> and <i>messageuid</i> , plus the values for the flags: <i>answered</i> , <i>deleted</i> , <i>draft</i> , <i>flagged</i> , <i>seen</i> , where each flag value can be <i>kTrue</i> (adds the flag), <i>kFalse</i> (removes the flag) or <i>kUnknown</i> (leaves the flag unchanged)
IMAPSubscribeMailbox	Adds a specified mailbox to the server's set of "active" or "subscribed" mailboxes as returned by IMAPListSubscribedMailboxes by issuing a SUBSCRIBE command to the server; requires <i>socket</i> and <i>mailboxname</i>
IMAPUnsubscribeMailbox	Removes a specified mailbox from the server's set of "active" or "subscribed" mailboxes as returned by IMAPListSubscribedMailboxes by issuing an UNSUBSCRIBE command to the server; requires <i>socket</i> and <i>mailboxname</i>

Multi-threading

The Web commands are now multi-threaded, when running on a multi-threaded Omnis Server. The Web commands allow another thread to execute in the multi-threaded server while the current command runs. Note that the same socket cannot safely be used concurrently by more than one thread.

Email Headers

In addition to the above enhancements, the SMTPSend and MailSplit commands now support international characters in email headers (using RFC2047). See the updated commands in the Omnis Help (press F1).

Secure Web Commands

Support for secure connections has been added to the existing package of web commands, including the HTTP, FTP, SMTP and POP3 client commands. In addition, there is a new set of commands to allow communications with an IMAP email server.

For further information about the new secure Web Commands please see the Omnis Help (F1) or the *Omnis Command Reference* manual which is available on the Omnis Studio DVD or to download from the Omnis website.

Functions

The following are new or updated Omnis functions.

cnubintolist()

Function group	Execute on Web Client	Platform(s)
List	NO	All

Syntax

cnubintolist(*binary*)

Description

Returns the list created by converting *binary* to a list, where *binary* was created in non-Unicode Studio by assigning a list to a binary variable.

Example

```
; Assign a list to a binary var created in a non-Unicode Studio
Calculate lBinary as lOldList
; Converts a binary variable to a list in a Unicode Studio
Calculate lNewList as cnubintolist(lBinary)
```

decxtea()

Function group	Execute on Web Client	Platform(s)
Binary Field	NO	All

Syntax

decxtea(*binary*,*key*)

Description

Returns the *binary* result of decrypting *binary* (previously encoded using `encxtea()`) with the binary *key*; the key must be 128 bits long.

DNet.\$addclass()

Function group	Execute on Web Client	Platform(s)
DNet	NO	All

Syntax

DNet.\$addclass(*FileName*)

Description

Loads classes from *FileName* (or FileNames if a comma-separated is provided).

DNet.\$basefolder()

Function group	Execute on Web Client	Platform(s)
DNet	NO	All

Syntax

DNet.\$basefolder()

Description

Returns the base folder of the .NET classes.

DNet.\$getenum()

Function group	Execute on Web Client	Platform(s)
DNet	NO	All

Syntax

DNet.\$getenum(*EnumName*)

Description

Returns the value of the fully qualified specified *enum* (e.g. System.IO.FileMode.Open).

encxtea()

Function group	Execute on Web Client	Platform(s)
Binary Field	NO	All

Syntax

encxtea(*binary*,*key*)

Description

Returns the *binary* result of encrypting binary using the eXtended Tiny Encryption Algorithm (XTEA) with the binary *key*;the key must be 128 bits long.

FileOps.\$setunixpermissions()

Function group	Execute on Web Client	Platform(s)
FileOps	NO	All

Syntax

FileOps.\$setunixpermissions(*cPath*,*cPermissions*)

Description

Sets the Unix permissions for the file identified by *cPath*. The parameter *cPermissions* has the same syntax as the permissions string returned by \$getunixpermissions().

getenv()

Function group	Execute on Web Client	Platform(s)
General	NO	All

Syntax

`getenv(name)`

Description

Returns the value of the Omnis process environment variable with the specified *name*.

Example

```
; Get the executable search PATH environment variable
Calculate lVar1 as getenv("PATH")
```

isclear()

Function group	Execute on Web Client	Platform(s)
General	YES	All

Syntax

`isclear(expression|fieldname)`

Description

Returns true if the *expression* or *field* has value NULL, zero (for numeric data types only) or empty.

JavaObjs Library.\$resetclasscache()

Function group	Execute on Web Client	Platform(s)
JavaObjs Lib	NO	All

Syntax

`JavaObjs Library.$resetclasscache()`

Description

Deletes the Java objects class cache file, allowing the Java objects library to rebuild it the next time Omnis starts up; any classes added with `$addclass()` will be lost from the cache.

listenv()

Function group	Execute on Web Client	Platform(s)
General	NO	All

Syntax

`listenv()`

Description

Returns a two column list of the Omnis process environment variables. Column 1 contains the variable name, and column 2 the variable value. The list is sorted on the variable name column (case insensitive).

Example

```
; Get the list of environment variables
; There is no need to define the list first; the returned list has
; 2 character columns, name and value
Calculate lList as listenv()
```

loctoutc()

Function group	Execute on Web Client	Platform(s)
Web Services	NO	All

Syntax

`loctoutc(datetime)`

Description

Converts the *datetime* (or time) from the local timezone to UTC (Coordinated Universal Time), and returns the result. (Only available with the Omnis Web Services product).

putenv()

Function group	Execute on Web Client	Platform(s)
General	NO	All

Syntax

`putenv(name,value)`

Description

Sets the Omnis process environment variable with the specified *name* to the specified *value*; creates a new environment variable if necessary; returns Boolean true for success, false for failure.

Example

```
; Set the environment variable MYVAR to the value MYVALUE  
Do putenv("MYVAR", "MYVALUE")
```

sys()

sys(215)

sys(215) is a new function. It returns the pathname of the folder containing the Omnis executable, including the terminating path separator. This is useful on Windows Vista where the Omnis executable may reside in a different folder to your application files.

sys(115)

On Windows Vista returns the pathname of the folder containing the installed writeable files, including the terminating path separator (usually a sub-folder of the AppData Local folder). To get the pathname of the folder containing the Omnis executable under Vista, use sys(215).

sys(130 .. 139)

sys() functions 130 to 139 are no longer valid since the old or so-called V2 DAMs are not supported in Omnis Studio 5.

sys(112) & sys(113)

sys(112) and sys(113) are no longer valid since Balloon Help and Publish and Subscribe are not supported in Omnis Studio 5.

tzcurrent()

Function group	Execute on Web Client	Platform(s)
Web Services	NO	All

Syntax

tzcurrent()

Description

Returns the short character identifier for the time zone of the current date and time. (Only available with the Omnis Web Services product).

tzdaylight()

Function group	Execute on Web Client	Platform(s)
Web Services	NO	All

Syntax

tzdaylight()

Description

Returns the short character identifier for the daylight saving time zone. (Only available with the Omnis Web Services product).

tzstandard()

Function group	Execute on Web Client	Platform(s)
Web Services	NO	All

Syntax

tzstandard()

Description

Returns the short character identifier for the standard time zone. (Only available with the Omnis Web Services product).

utctoloc()

Function group	Execute on Web Client	Platform(s)
Web Services	NO	All

Syntax

utctoloc(*datetime*)

Description

Converts the *datetime* (or time) from UTC (Universal Coordinated Time) to the local timezone, and returns the result. (Only available with the Omnis Web Services product).