What's New in Omnis Studio 3

October 2000

Table of Contents

ABOUT THIS MANUAL....4

CONVERTING STUDIO 2.X APPS....4

OMNIS WEB CLIENT....5

MULTI-THREADING....5
Introduction....5
Multiple Method Stacks....5
Using the Server....6
Database Access....6
Method Commands....7
LOAD SHARING....9
Introduction....9
Setting up load sharing....10
Load Sharing Mechanism....11
Using version 2.x web clients....12
CLIENT METHOD EXECUTION....12
MULTIPLE FORMS....13
OTHER WEB CLIENT ENHANCEMENTS....14

IMPROVED APPLICATION DESIGN TOOLS....15

COMPONENT STORE....15
METHOD EDITOR....15
OTHER MODIFICATIONS....16

MISCELLANEOUS FEATURES....17

About This Manual

This document describes the new features and enhancements in Omnis Studio 3. This version has many new features for the Omnis Web Client, support for new operating systems and improvements to the IDE. Additional info is found in the Readme.txt.

This manual is for existing Omnis Studio version 2 users only.

Converting Studio 2.x apps

IMPORTANT NOTE: Omnis Studio 3 will convert Studio version 2.x apps automatically. After you have opened existing Studio libraries in Studio 3 they will no longer work with Studio 2. Therefore you should make secure backups of all the libraries you wish to convert to Omnis Studio 3 before opening them in Studio 3.

Omnis Web Client

Multi-threading

Introduction

In Studio 3 the Omnis Server allows multiple requests to be processed allowing smoother allocation of available processor time and avoiding any lengthy delays on the client. To handle these multiple requests the Omnis Server is now multi-threaded.

The new multi-threaded server (MTS) maintains a pool of method stacks that can process web client requests simultaneously. The pooling mechanism allows a balance to be struck between performance and server resources - the number of method stacks in the pool is configurable as the $root.$prefs.$serverstacks property.

Multiple Method Stacks

The standard Studio program contains a single method stack. This means that (broadly speaking) once a method call has been pushed onto the stack no other method call can begin to execute until the first call has completed. In contrast the MTS contains a pool of method stacks which are available to process web client requests. When a request to execute a method is received from a web client that method call is pushed onto any unused stack or, if there are no unused stacks, the message is queued until one becomes available. Each method stack runs in its own thread which means that if a method stack is stalled (for example it is waiting for the database server) the other stacks will continue to execute.

Typically the MTS has to cope with a very large number of simultaneous clients each with their own remote form and remote task instances. Of course in typical cases only a small proportion of clients will require the use of the server at any one time. In any case multi-threading does not increase the server processor time available, it just allows the available processor time to be allocated in a smoother way. The method stack pool mechanism allows a balance to be struck between performance and server resources - the number of method stacks in the pool is configurable as the $root.$prefs.$serverstacks property.

This means that a remote client will not get the same method stack every time it executes a method on the server. Each method stack contains its own state which, apart from during an individual method call, does not belong to any particular client. This state includes CRBs for all files and variables (apart from class variables) and such modes as the current list. A client cannot rely on any of this state being preserved across different method calls, the only thing belonging to the client are its instance and task variables. So a client must do such things as setting the main file and current list each time one of its methods executes, and

should not rely on such things as the values of memory-only fields being maintained across method calls. As a special case the class variables for the remote task and form classes are shared amongst all clients so can be used to hold shared data (see below for the warnings about the care needed when using shared variables).

Using the Server

When the MTS starts up it open's the libraries, datafiles and SQL session pools required by the clients (see below for description of a SQL session pool). The Start server command is then issued to cause the method stacks and associated threads to be created and to cause Studio to start to listen for web client requests. The Start server command can specify an optional stack initialisation method; when specified this method is pushed onto every client method stack and allowed to execute (so if $serverstacks is 5 it will execute five times), so it can be used to initialise the state of the method stacks. The Start server command generates a fatal error if due to lack of resources or some other reason it is unable to complete successfully.

When the server is to be stopped it is recommended the Stop server command is issued but quitting the Studio program achieves the same results.

When the server is active it continues to be responsive to local events and could, for example, be displaying a window with 'Start server' and 'Stop server' buttons. It is not recommended that the server program performs any substantial tasks when it is listening for client requests.

Any runtime errors generated by client methods are reported in the trace log (using a similar mechanism as errors during library conversion), but this default behaviour can be overridden by making sure each client method stack has an error handler. The stack initialization method call for the Start server command can be used to define an error handler for each method stack.

Database Access

There is a completely new DAM interface for Studio 3 that supports multi-threading on the Omnis server. This means there will be new DAMS for Studio 3.0, including ODBC, ORACLE, DB2 and SYBASE.

The new DAMs are implemented as non-visual external components. The new DAMs will not work with the current 4GL SQL commands, rather you need to use object variables based on the new non-visual DAMs, and interact with a DAM using the methods of the object. Using the non-visual approach an application creates object variables of a particular DAM class which are instantiated when the object is created by the component. There is a group of common methods which apply to all DAM objects and a set of DAM specific methods based on the type of object.

Database sessions

There are new multi-threaded DAMs available. These new DAMs are non-visual components that are instantiated in object variables to become the equivalent of sessions.

Tables and schemas

The table class has been revised to support the new DAMs. Tables that use earlier DAMs will function in Studio 3.0 but not in a client method running on the MTS.

Current DAMs

The earlier DAMs and commands (including such concepts as the current session) are supported in 3.0 for backward compatibility. However, the old DAMs are not safe in a multi-threaded environment so the SQL commands cannot be used in a client method running on the MTS.

Omnis SQL

Note that there is no multi-threaded DAM for Omnis SQL. This means that Omnis SQL cannot be used with MTS.

Native datafile access

There should be no problems with using the native datafile commands with the MTS. The list of open datafiles is shared amongst all the method stacks and all the datafiles which might be used by the clients should be opened before the Start server command is issued.

Each method stack has its own set of CRBs for the read/write, read-only and memory-only files and also has its own main file, find tables, and the various modes used by the native datafile commands. Since these belong to a method stack and not a client it is important that client methods do not make any assumptions about the modes, contents of the CRBs, etc. between method calls. In particular there is no way that a client can assume that a find table exists between method calls. Of course these restrictions also apply to single threaded Studio when it is acting as a web client server.

Method Commands

There is a new Threads… command group which contains the following commands:

Start server

This command is used to create the client method stacks and associated threads and start the thread which listens for client requests. It takes an optional Stack initialization method as a parameter as described under 'Using the Server' above. The flag is cleared if used in a single threaded Studio or the serial number does not allow clients to connect. A fatal error is generated if for some other reason it is not possible to create the stacks and threads and start the listener.

Stop server

This commands stops the server from responding to client requests. Once the server has been started it is recommended it is stopped before quitting the Studio program, before using the Studio program for anything apart from serving client requests, and before opening or closing any datafiles or libraries.

The Stop server command disposes of all remote task and form instances. The resources used by the client stacks and threads are not released but they will be reused by the next Start server command.

Begin and End critical block

These commands are used to denote a section of code which needs to execute in single threaded mode without allowing other client methods to execute. For example:

```
Set current list cList
Begin critical block
   Build list from file
End critical block
```

Here cList is a class variable which is shared amongst the clients and the critical block is used to prevent other clients from accessing the list whilst it is partly built. Generally class variables should only be used when the shared functionality is essential and only with care:

```
Calculate cString as 'abc' ;; OK
Calculate cString as $cinst.$xyz()
   ; only OK inside a critical block
```

Simple atomic operations such as the first line are safe, but when a method call is involved it may be interrupted by other threads and cause problems. Class variables should not be as bind variables or as the return list for SQL operations.

Yield to other threads

This command is a hint that the executing thread is waiting for other threads and is prepared to yield its processor time. It can be used when waiting for semaphores (since with the MTS another client stack could be holding the semaphore):

```
Do not wait for semaphores
Repeat
   Prepare for edit
   If flag true
      Break to end of loop
   End If
   Yield to other threads
Until break
```

Commands which are not available to a client

The following commands are not available for client methods running on the MTS. They usually generate a ' Command not available when executing a client method' fatal error but some (such as the Debugger… group) simply do nothing:

```
Start server
Stop server
Enter data
Prompted find
The Libraries… group
The Classes… group
The Logon… group
The Sessions and cursors… group
The Select table… group
The Fetch… group
The SQL scripts… group
The Client import files… group
Open data file
Prompt for data file
Create data file
Close data file
Open lookup file
Close lookup file
The Data management… group
The Message boxes… group
The Debugger… group
Quit Omnis
```

Any other command which would cause a dialog to be displayed on the server are not available for client methods running on the MTS.

Similarly the notation for opening and closing libraries and datafiles is not functional in a client method. There are of course many other commands and notation which does not make sense to try to use from a client method.

Load sharing

Introduction

Load sharing allows a pool of multi-threaded Omnis server processes, running on one or more machines, to serve clients. Once a client connects to an Omnis server process, all subsequent requests for that client need to be handled by the same Omnis server process. Therefore, load sharing provides a mechanism that assigns a new client connection to an

Omnis server process. Load sharing is implemented for Win32 and Linux Omnis servers only.

The OmnisServer property in an HTML page currently has the syntax [(IP address|domain name):](service name|port number). To use load sharing, you prefix this property with a name for the pool of Omnis server processes and a comma, for example “Omnis,6000”, or “Omnis,194.131.70.197:6000”. In this case, the address information in the property no longer addresses an Omnis server. Instead, it addresses a new module, a load sharing process.

When a new connection arrives at the Web Server plug-in (ISAPI, CGI or Apache module located in the cgi-bin), the plug-in recognises the new syntax of the OmnisServer property. If it is prefixed by a pool name, the plug-in connects to the load sharing process, and sends it a message that asks for the address of a server process in the pool.

The load sharing process typically returns the address and port number of the least busy process in the pool.

The plug-in then connects to this process, and sends the web client connection to it. When the plug-in responds to the client, it includes the address of the Omnis server process in the response.

When the client sends subsequent messages to the Web Server for this web client connection, it sends the address returned in the connect response instead of the OmnisServer property. Thus the only additional overhead imposed by load sharing occurs during connection setup.

So how does Omnis know (1.) which Omnis server processes exist, and (2.) which Omnis server process is the least busy? The load sharing process (LSP) has an .ini file, which contains the pool names for the pools for which it is responsible, and for each pool, the addresses of the Omnis server processes in the pool. Periodically, the load sharing process polls each Omnis server process, and asks it for the percentage of Web Client connections currently in use (using the serial number as the maximum), and if available, an estimate of its CPU utilisation. The load sharing process combines this information to determine which process is the least busy.

The time interval between polls of each Omnis server process is configurable via the .ini file. Typically, once every 10 or 20 seconds is frequent enough.

Setting up load sharing

Note Studio 3.0 and older clients can connect to a load sharing process (LSP).

On the www server

The html file will have the 'OmnisServer' property changed to the LSP, for example:

```
OmnisServer="Omnis,6001"   or
OmnisServer="Omnis,123.456.789.010:6001"
```

Where Omnis is the name of a pool of Omnis server processes and 6001 is it's port number.

The LSP would typically be running on the www server. This is a single executable omnislsp.exe. A configuration file (omnislsp.ini) must be made to accompany it, this takes the following format:

```
[Setup]
Port=6001
QuietMode=0
BucketSize=100
LogLineThreshold=16
Pool1=Omnis
[Omnis]
PollTimer=10
Server1=123.145.71.123:7001
Server2=123.145.71.124:7002
```

The commands for the lsp are: omnislsp -start
omnislsp -stop

(With the omnislsp.ini in the same directory)

On the LSP servers

The servers specified in the omnislsp.ini do not run any www server software. The servers may be stopped and restarted without the need to stop the LSP. More information is available in the Studio 3.0 functional specification.

Load Sharing Mechanism

The load sharing process periodically polls the processes in a pool of Omnis Server processes. Each server returns the following information:

- The current number of connections to the server
- The maximum number of concurrent connections allowed to the server (specified by the serial number)
- The number of requests (mainly events or new connections) received since the last poll
- The total elapsed time in milliseconds taken to process the requests

The load sharing process organizes the servers into groups, based on the results of the information returned from polling the servers. It maintains a configurable number of groups (N), numbered 0 to (N-1). Using a bucket size (B milliseconds), and the average time to process a request (T), calculated from the results of the most recent poll, a server is in group G if and only if:

G*B <= T and T < (G+1)*B

One exception is that if T > N*B, then the server belongs to group N-1.

Thus, the servers are organized into groups based on how busy they are.

When a connection request arrives at the load sharing process, it allocates a server to the request as follows. It traverses the groups, from 0 to N-1, looking for a server that has some free connections. Within a group, it looks for the server with the smallest percentage of connections in use, using the results of the last poll. If there is more than one server with the same smallest percentage of connections in use, then the process allocates the connection to the server to which it least recently allocated a connection. At this point, the load sharing process also updates the connection statistics from the last poll, to reflect the new connection. The traversal stops when a free process has been found.

Using version 2.x web clients

A 2.4 client or earlier will receive an error if it tries to connect to a 3.0 Omnis server, set up for load sharing. If you want 2.4 clients to be able to connect to 3.0 Omnis servers using load sharing (perhaps to enable the 2.4 client to be updated to 3.0), you can do so by specifying an additional server address in the OmnisServer property in your HTML page containing the web client plug-in. For example:

OmnisServer = "OMNIS,194.131.70.122:6000;194.131.70.122:5912"
or
OmnisServer="OMNIS,6000;5912"

In the case where there are two addresses in the OmnisServer property, separated by a semicolon, the first address will be used by 3.0 and later clients, and the second by 2.4 and earlier clients. The address for 2.4 clients must directly address an Omnis server, that is all load sharing parameters are irrelevant and should not be included.

Client method execution

You can specify that a remote form method is to run on the client rather than the server. Right-click on the remote form's method name and select the **Execute On Web Client** pop-up menu option.

This avoids the overhead of sending event messages between the client and server and reduces the workload on the server.

You should only allow simple methods to be executed on the client, as the web client libraries need to be kept as small as possible and the amount of information (such as variable values) available on the client is limited.

Multiple forms

You can now open more than one form within a single web client connection, that is, within a single remote task instance. At any one time, only one of these multiple instances is visible, and the forms must be from the same library.

Studio 2.2 introduced the $changeform() method of a remote task instance. Studio 3.0 contains two further remote task instance methods, $openform() and $closeform(). Like $changeform(), both these methods take a single argument, the form name.

If the form passed to $openform() already has a remote form instance open in the context of the current task instance, then it becomes the visible form for the current task. Otherwise, Omnis constructs a new instance of the remote form in the current task, and makes the new remote form instance the visible form. This behavior is analogous to the $openonce() method of a window class.

The $closeform() method destructs the remote form instance for the named form. It is possible to close the last open remote form instance, but this is not recommended. If the referenced form is not visible, the client observes no affect; otherwise, the most recently visible open remote form instance becomes visible.

There are some further restrictions to note:

- $closeform, $openform and $changeform cannot be used in the constructor or destructor of a remote form instance or remote task instance. If used, OMNIS generates a runtime error.
- Multiple calls to $openform or $changeform during the processing of a single event will result in only the last call to $openform or $changeform having any affect.
- Calling $showurl or $showmessage in the destructor of a remote form has no affect.
- All forms must be in the same library.

You should use task variables to handle communication between multiple remote form instances in a remote task instance.

To facilitate communication between different remote form instances, remote forms can also receive one event, evFormToTop. In design mode, you can enable this event for a form, using the $events property of the form. The event generates a call to $event in the class methods group of the form. evFormToTop occurs when an existing remote form is about to become visible on the client as a result of a call or calls to $openform or $closeform.

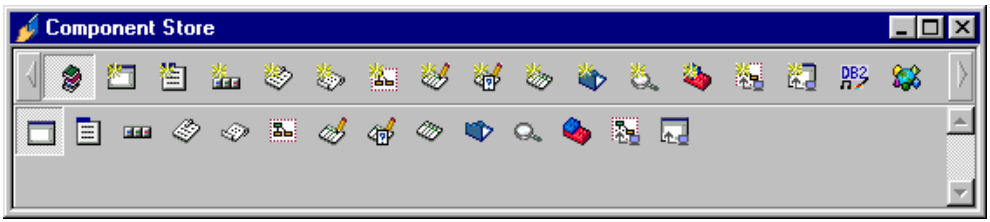
Other Web Client enhancements

- A headed list box property allowing you to hide the heading
- Remote form properties providing Enter and Cancel key functionality
- An ActiveX/Plug-in property that speeds up secure connections by identifying the secure URL of the OmnisAPI/CGI
- New controls including Icon Array, Marquee, Animated GIF and a background line object
- An enhanced $showurl, allowing you to display a browser window with no title
- A mechanism to update the core client files
- Ability to print reports directly (Windows and Mac only)

Improved application design tools

Component Store

The Component Store now defaults to an industry-standard label-free palette:



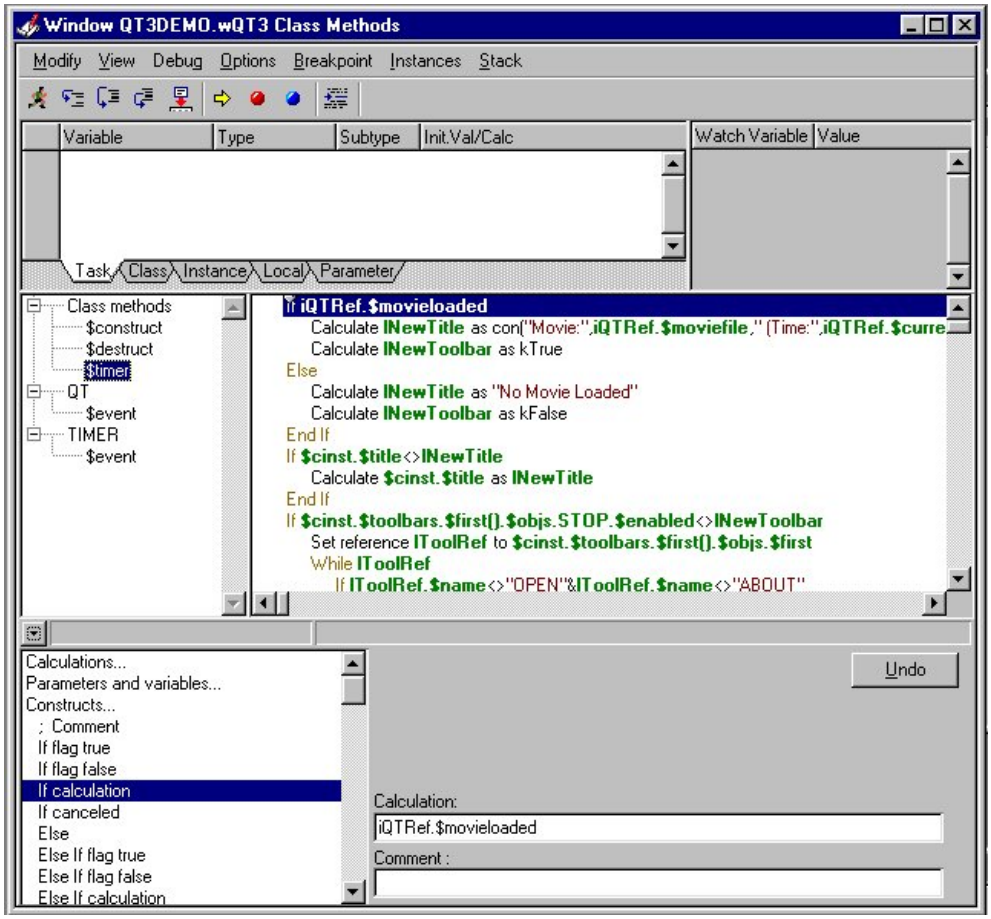
Each icon has a tooltip explaining which group of components it displays or which wizard/template it loads.

This new design makes it quicker to locate the wizard or template you want.

Method Editor

The method editor now has hidden toolbar droplists/popup menus, plus a versatile tree-style method list.

Should the function list be hidden, the currently method of selection via the toolbar droplists/popup menus will be available.



Other modifications

- The automatic notation completer provides descriptive tooltips and multiple choices where available
- A find-and-replace class notation method, to support automated translation tools
- The Icon Editor supports true color

Miscellaneous features

- Omnis Studio 3 supports some Windows 2000 features such as the new style file dialogs.
- A timeout field lets users specify how many seconds Omnis should wait when writing to or reading from a port
- Mac USB support for serial devices
- A modified Port Parameters Structure (PRIportparms) and Port Destination Structure (PRIdestParmStruct) for all platforms
- Clipboard support for external objects
- Report display enhancements

How to use this manual

The on-line documentation is designed to make the task of identifying and accessing information about Omnis Studio as easy and intuitive as possible.

You can navigate this PDF document, or find topics, in a number of different ways.

Bookmarks

Bookmarks mark each topic in a document. To view the bookmarks in this document, click on the Bookmark icon on the Acrobat toolbar or select the **View>>Bookmarks and Page** menu item. In Acrobat Reader 4, you can click on the Bookmarks tab.

Click on an arrow icon  to open or close a topic, and click on a topic name or double-click a page icon  to move directly to a topic.

Thumbnails

Thumbnails are small images of each page in the document. To view the Thumbnails in this document click on the Thumbnails button on the Acrobat toolbar, or select the **View>>Thumbnails and Page** menu item. In Acrobat Reader 4, you can click on the Thumbnails tab.

You can click on a thumbnail to jump to that page. Also you can adjust the view of the current page by moving and/or sizing the gray page-view box shown on the current thumbnail.

Browsing

You can use the Browse buttons on the Acrobat toolbar to move back and forth through the document on a page by page basis. You can also click on the **Go Back** to return to your last view or location.

Find

You can find a text string using the **Tools>>Find** menu item. To find the next occurrence of the text you can use the **Tools>>Find Again** option. If you reach the end of the document, you can use the Ctrl-Home key to go to the beginning and continue your find.
See also Search (on the next page of this guide).

Search

If you have the Acrobat Search plug-in (available under the **Tools>>Search** menu in some versions of Acrobat Exchange and Reader), you can use the Studio Index to perform full-text searches of the entire Omnis Studio on-line documentation set. Searching the Studio Index is much faster than using the **Find** command, which reads every word on every page in the current document only.

To Search the Studio Index, select **Tools>>Search>>Indexes** to locate the Studio index (Index.pdx) on the Omnis CD. Next, select **Tools>>Search>>Query** to define your search text: you can use Word Stemming, Match Case, Sounds Like, wildcards, and so on (refer to the Acrobat Search.pdf file for details about specifying a query). In the Search Results window, double-click on a document name (the first one probably contains the most references). Acrobat opens the document and highlights the text. To go to the next or previous occurrence of the text, use the Search Next or Search Previous button on the Acrobat toolbar.

Grabbing Text from the Screen

You can cut and paste text from this document into the clipboard using the Text tool. For example, you could copy a method or code snippet and paste it into the Omnis method editor.

Getting Help

For more information about using Acrobat Reader see the PDF documents installed with the Reader files, or select the **Help** menu on the main Reader menu bar.