OMNIS Studio 2.4 Update

OMNIS Software

February 2000

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

© OMNIS Software, Inc., and its licensors 2000. All rights reserved.
Portions © Copyright Microsoft Corporation.

OMNIS® is a registered trademark and OMNIS 7™, OMNIS Studio™, and OMNIS Web Client™ are trademarks of OMNIS Software, Inc.

Microsoft, MS, MS-DOS, Visual Basic, Windows, Windows 95, 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, AppleTalk, and Macintosh are registered trademarks and MacOS, Power Macintosh and PowerPC are trademarks of Apple Computer, Inc.

Linux is a trademark of Linus Torvalds.

IBM and AIX is a registered trademark and OS/2 is a trademark of International Business Machines Corporation.

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

Sun, Sun Microsystems, the Sun Logo, Solaris, Java, and Catalyst are trademarks or registered trademarks of Sun Microsystems Inc.

HP-UX is a trademark of Hewlett Packard.

OSF/Motif is a trademark of the Open Software Foundation.

Acrobat is a trademark of Adobe Systems, Inc.

pORACLE is a registered trademark and SQL*NET is a trademark of Oracle Corporation.

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

INFORMIX is a registered trademark of Informix Software, Inc.

EDA/SQL is a registered trademark of Information Builders, Inc.

CodeWarrior is a trade mark of Metrowerks, Inc.

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

Table of Contents

About This Manual

This document describes the new features in OMNIS Studio 2.4 and the enhancments added in Studio version 2.1, 2.2, and 2.3.

The information in this manual supplements that provided in the *Using OMNIS Studio* manual, *OMNIS Programming*, and the *OMNIS Web Client* PDFs. In addition, it contains some errata, and a reference section documenting new Web commands.

OMNIS Studio 2.4

OMNIS Studio 2.4 includes the following new features.

Getting Started Manual

There is a new manual called *Getting Started with OMNIS Studio*. This book is aimed at the first-time OMNIS Studio user, and provides the basic information and practical exercises to get you up and running with OMNIS Studio. We recommend you work through this book before reading the other manuals.

App Builder

The new OMNIS App Builder lets you build complete OMNIS applications from start to finish. Its wizard-like environment lets you choose what you want to store in your database or application, exactly how it will look, and it lets you decide what type and style of windows you want your application to have. You can run the App Builder from the Welcome application, available when you startup OMNIS, or from the main Tools toolbar.

The App Builder provides many different application templates for business and home use, including invoices, contacts management, and many more. The templates in the App Builder will be constantly updated; in fact, while you step through the App Builder it lets you download any new or updated application templates from the OMNIS ftp site.

Oracle 8

Support for Oracle 8 has been added under Windows 32 only. The Oracle8 DAM, named doracle8.dll, has most of the existing functionality of the Oracle7 DAM and works in a very similar way. The Oracle8 DAM has some added functionality to support the LOB datatypes as well as improved performance, and supports Net Client v8.1.5 clientware. The new DAM can be used against an Oracle7 server, using the recommended clientware, but you will encounter the restrictions described below, mainly concerned with data type mapping of large objects.

Logon

When logging on to an Oracle 8 server, the method is slightly different from previous Oracle DAMs including the Oracle7 DAM. To log onto Oracle 8, the method is now:

```
Start session {doracle8}
Set hostname {dbora8}
Set username {scott}
Set password {tiger}
Logon to host (Omnis Character Set)
```

This differs from Oracle7 in that you do not have to specify any prefixes to the hostname e.g. “@” and also the username and password can be specified in the relevant command rather than as part of a connection string specified in the *Set username* command. This logon procedure is more consistent with the procedures for other DAMs.

Server-specific Programming

The Oracle8 DAM does not contain any new server specific keywords. All of the keywords that were available in the Oracle7 DAM are still available in the new DAM and the functionality is the same.

New Datatypes

CLOBs, NCLOBs, and BLOBs

CLOBs, NCLOBs and BLOBs are new Oracle8 datatypes that deal with large objects. Internal Large Objects or LOBs (specifically BLOBs, CLOBs, and NCLOBs) are stored in the database tablespaces in a way that optimizes space and provides efficient access. These LOBs have the full transactional support of the database server. The maximum length of a LOB/FILE is 4 gigabytes. Internal LOBs have copy semantics. Thus, if a LOB in one row is copied to a LOB in another row, the actual LOB value is copied, and a new LOB locator is created for the copied LOB.

The Oracle8 DAM uses locators to point to the data of a LOB or FILE. These locators are transparent as the DAM performs operations on the locator to insert, update, delete and

fetch the values. This means that you only ever deal with the values of the lobs and not the locators.

If you're interested in working with the locators rather than just the values you can do using PL/SQL in conjunction with the dbms_lob package. You'll find more information in the Oracle8i Supplied Packages Reference.

BFILES

External LOBs (FILEs) are large data objects stored in the server's operating system files outside the database tablespaces. FILE functionality is read-only. Oracle currently supports only binary files (BFILEs). The Oracle8 DAM uses locators to point to the data of a FILE. The FILE locator will be transparent to the user, as the DAM will return the value of the external file and not the locator when performing transactions with the BFILE datatype. Even though the BFILE datatype is read-only you can insert a directory alias and filename into the column. These values are assigned to a single OMNIS binary variable and separated by the '&' symbol. The DAM will assign these values to the locator so that when a fetch is performed on the locator the binary representation of the external file corresponding to the alias and filename will be returned. An example is shown below.

```
; A Directory alias needs to be created on the server
; that points to an OS folder
Perform SQL {create or replace directory sound as 'c:\bfiles'}
; BFILE1 and BFILE2 are OMNIS variables of type Binary.
; The variable is calculated as
; '<DirectoryAlias>&<Filename>'.
Calculate BFILE1 as 'sound&wav2.wav'
Calculate BFILE2 as 'sound&wav3.wav'
Perform SQL {insert into bfiletest values(1,@[BFILE1],@[BFILE2])}
; We can now select the data back and this time we will
; receive the binary representation of the file. Wav1 and
; Wav2 are OMNIS variables of type Binary
Perform SQL {select * from bfiletest}
Fetch next row  into { #1,Wav1,Wav2 }
; The values in contained in Wav1 and Wav2 can now be written
; to the local drive using the OMNIS Fileops commands
Calculate File as 'c:\windows\desktop\wavtest.wav'
Do Fileops.$createfile(File) Returns lErr
Do Fileops.$openfile(File) Returns lErr
Do Fileops.$writefile(Wav1) Returns lErr
Do Fileops.$closefile() Returns lErr
```

Datatype Mapping

Omnis to Oracle

Omnis Datatype	Oracle8 Datatype
Character/National <=2000	VARCHAR(n)
Character/National >2000	CLOB
Short date (all subtypes)	DATE
Short time	DATE
Date time(#FDT)	DATE
Short Integer(0-255)	NUMBER(3,0)
Long Integer	NUMBER(11,0)
Short Number 0dp	NUMBER(10,0)
Short Number 2dp	NUMBER(10,2)
Number Floating dp	FLOAT
Number 0..14dp	NUMBER(16,0..14)
Boolean	VARCHAR2(3)
Sequence	NUMBER(11,0)
Picture	BLOB
Binary	BLOB
List	BLOB
Row	BLOB
Object	BLOB
Item Reference	BLOB

Oracle to Omnis

Oracle Datatypes	Omnis Datatypes
CHAR	Character
VARCHAR2	Character
CLOB	Character
NCLOB	Character
BLOB	Binary
BFILE	Binary
LONG	Character
RAW	Binary
LONG RAW	Picture
DATE	Date time (#FDT)
NUMBER	Number floating dp
NUMBER(10)	Number floating dp
NUMBER(10,8)	Number floating dp
NUMBER(18)	Number floating dp
NUMBER(16,15)	Number floating dp
NUMBER(18,15)	Number floating dp
FLOAT	Number floating dp
FLOAT(30)	Number floating dp

Troubleshooting

Version restriction

The Oracle8 DAM is restricted to users of OMNIS Studio 2.0 and later versions. An attempt to use the DAM with any prior version will result in a runtime error.

LONG datatypes

Long, Long Raw, and Long Varchar datatypes can no longer be used fully with the Oracle8 DAM. Whilst any existing data in these columns can still be selected, fetched and mapped to binary variables, the DAM no longer inserts bind variables into columns of these types. This is because any bound character or binary data larger than 2000k will be mapped to a CLOB or BLOB type.

Transaction mode for new datatypes

When performing transactions that use the new LOB datatypes the transaction mode must be set to either Automatic or Generic. This is because lob and file locators cannot be used across transactions. The DAM performs functionality on these locators and when the

transaction mode is either Automatic or Generic, the DAM controls when to commit the command, which would be after all the LOB functionality has been performed. When the transaction mode is Server, Oracle commits (or rollbacks) after every statement and any LOB functionality performed by the DAM would result in an error.

OMNIS Studio 2.1/2.3

This section contains the features and enhancements added to OMNIS Studio 2.1 and 2.3.

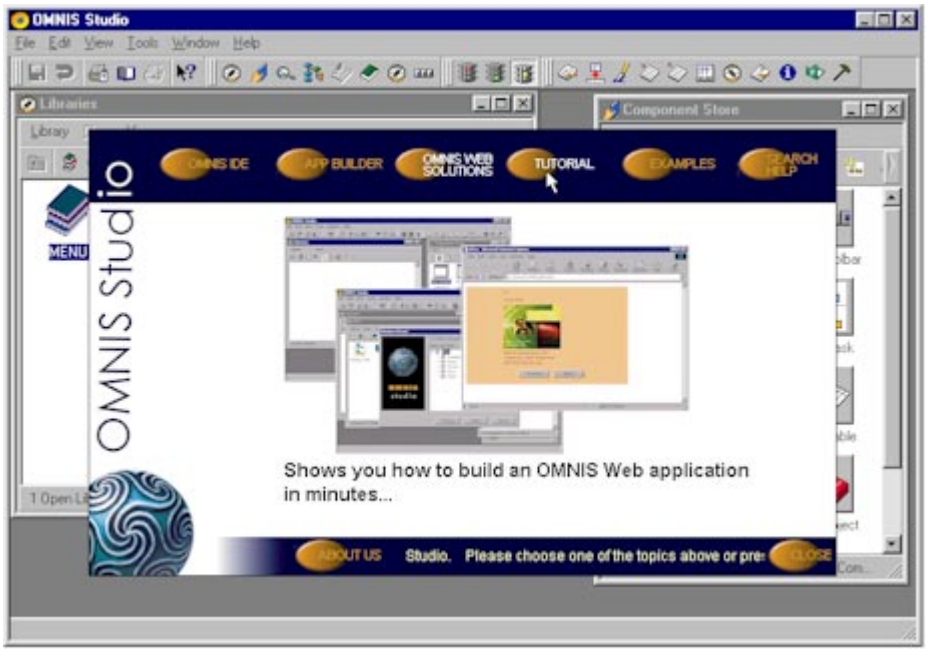
Design Environment

New Welcome Application

OMNIS Studio 2.2 and later has a new Welcome application that (when enabled) appears when you first launch OMNIS. The new application has an entirely new interface, using several of the new components, and has many new examples. The new Welcome library also contains a completely new tutorial that uses the OMNIS Web Client and shows you how to build an OMNIS web application in minutes.

The About screen in the Welcome application uses several of the new external components available in OMNIS Studio 2.2 and later, including the new Roll Button. Most of the browsers and help windows in the Welcome library use the new Document Viewer for displaying Html pages and JPEG pictures.

The new Welcome application contains many topics for beginners and experienced OMNIS users alike. Amongst its many new features, the Welcome app has a **New Users** section that contains information about OMNIS and lets you access the new Tutorial. The **Knowledge Store** lets you search the OMNIS help files and technical notes.



The **New Users** section contains information about the OMNIS IDE, the **App Builder** lets you create simple home and business applications from scratch, **OMNIS Web Solutions** describes how you can create web applications using the OMNIS Web Client, and the **Examples** section contains many new examples showing the wealth of design features and components available in OMNIS.

New Tutorial

The new tutorial is available under the New Users section in the Welcome application. To run the tutorial, start OMNIS and click on the **New Users** button in the Welcome About screen. You can read each step in the tutorial and then complete each task by placing the tutorial window to the right side of your screen.

The **Tutorial** shows you how to create an OMNIS web application using the OMNIS Web Client. The sample OMNIS database and finished library are available in the Welcome/Tutorial folder.

File menu

The File menu in the development version of OMNIS now has **New...** and **Open...** commands for creating and opening libraries.

Property Manager

The Property Manager context menu has two new items **Copy properties** and **Paste properties**.

– **Copy properties**
is enabled when a single object's properties are being displayed in the Property Manager. The option copies a list of the non-inherited properties of the object to the clipboard.

– **Paste properties**
is enabled when there are copied properties on the clipboard. If you select it, it assigns the properties listed on the clipboard to the objects currently displayed in the Property Manager.

These option let you transfer the properties of one object to another, or change the type of a window object, by creating a new object, copying the methods and the properties from the old object to the new object, and deleting the old object.

Delete Unused Variables

The context menu of the Variable pane in the method editor has a new item **Delete Unused Variables...**, available by Right-clicking on the variable pane away from a variable line. When selected, it opens a dialog from which you can select variables to delete. The dialog

displays the variables of the current type displayed in the variable pane, which are *potentially unused*. This means the variables could still be in use, for example, they could still be used in subclasses or notation.

Window Design Mode Sizing

If you hold down the Option/Alt key while resizing a design window, or a container field, you disable floating of the fields it contains, for the basic floating types (all floating types except the kEFposn... values; kEFposn floating still needs to occur, since this ties a field to a particular location).

Report Lines in Design Mode

Report design mode now allows easier manipulation of report objects, via the ability to insert and delete lines of objects in the report. This feature restores functionality that was available in the OMNIS 7 report editor via Ctrl/Cmnd-clicking.

Ctrl/Cmnd-click selects a line of objects, if you click on the report window background rather than an object. The line highlights where there are no objects. If at this point, you hold down the mouse and drag, you select a range of lines. Alternatively, after selecting a line, ctrl/cmnd-shift click on another line will select the range of lines between the 2 clicks.

When you select a line, selected objects become deselected. Similarly, when you select an object, the selected line(s) become deselected.

There are two new toolbar controls on the report design mode window:

– **Delete Selected Lines**
  deletes all objects whose start position is on a selected line - this includes positioning sections which are included in the range of objects selected, but excludes other sections. The lines which contained these objects are removed, meaning that lines below shift up.

– **Insert Line**
  inserts an empty line before the first selected line.

Notation Inspector

The Notation Inspector now has a "move root up one level" item on the toolbar and menu. This allows you to navigate up the tree starting from a point set using the magnifying glass control.

IDE Tools

Adhoc Reports

The adhoc reports tools menu and toolbar items are now toggles, which install and remove the adhoc reports menu.

Totals Only Report

You can now produce an Adhoc report of field totals only by selecting ‘Print totals only’ from the context menu. The report will only show totals for fields that have the ‘tot’ checkbox checked.

Custom SQL

You can now customize the SQL automatically generated for an Adhoc report, using the ‘Show SQL Statement’ context menu item. Enter the custom SQL into the ‘Custom SQL’ pane; a context menu is available for pasting in the default SQL. To build the report using the custom SQL check the ‘Build Report Using Custom SQL’ checkbox and press ‘Rebuild Output’ on the main ADHOC toolbar.

Add-on tools

There is a new Tools menu and toolbar item, for add-on tools. These are additional tools implemented as OMNIS libraries, which are present in the Startup folder of the development version. The add-on tools item is a hierarchical menu/toolbar pop-up menu, which contains an entry for each add-on tool.

You can implement your own add-on tool, by doing the following, and then placing the library in the startup folder:

Install the tool in $construct of the Startup_Task, by calling:

```
$root.$modes.$installtool(kEnvToolAddOn, <Menu line text>,
   <icon id>, <Menu line help bar text>)
```

For example:

```
$root.$modes.$installtool(kEnvToolAddOn,"&My Editor...",1710,
   "Opens my editor")
```

The icon id can be for an icon in #ICONS, or omnis/userpic, and is the icon used for the menu line for the tool.

The $installtool() method returns a non-zero number for success. The tool can use this number as the first of 200 possible configuration item numbers in omnis.cfg, accessed using the configuration notation methods described below. The returned number will be the same each time the tool installs itself, provided it does not change the <Menu line text>; if you delete omnis.cfg, you may get a different number, but this does not matter, since you've also lost any configuration information the tool may have stored.

When the user selects the add-on tool from the menu, OMNIS calls the method $exectool() in the tool's startup task. This typically opens a window, or brings it to the front.

There are a number of methods that let you save the window setup for your tool, or save other information. The methods are:

– $root.$prefs.$getconfiglong(kEnvToolAddOn,item,defaultValue)
  Returns the long integer configuration value <item>. If item is not present in omnis.cfg, returns defaultValue.

– $root.$prefs.$setconfiglong(kEnvToolAddOn,item,value[,dosave])
Sets the long integer configuration value <item> to <value>. If dosave is true, it saves omnis.cfg after setting the value; the default is not to save.

– $root.$prefs.$getconfigstring(kEnvToolAddOn,item,defaultValue)
Gets the string configuration value <item>. If the item is not present in omnis.cfg, returns defaultValue.

– $root.$prefs.$setconfigstring(kEnvToolAddOn,item,value[,dosave])
Sets the string configuration value <item> to <value>. If dosave is true, it saves omnis.cfg after setting the value; the default is not to save.

Tool Configuration

There is a new IDE option, accessed via **Tools>>Options**, called "idetools", on the General tab of the Property Manager.

If you click on the drop-down button, OMNIS displays a checkbox list, with an entry for each library that has installed itself as a tool. This includes the new add-on tools described above, and tools such as the VCS and SQL Browser. In the development version, these are usually all or most of the libraries in the Startup folder. You can check or uncheck items in the list, which opens or closes the appropriate library (and installs or removes its menu and toolbar entries) when you click off the checkbox list or hit return. OMNIS remembers the selected libraries in omnis.cfg, meaning that tool libraries in the startup folder only open at startup if they are checked. This saves both startup time, and resources (memory, file handles, buffers etc), since it enables you to select only the tools you will use.

Note that this is not an assignable notation item, and it only affects the development version of OMNIS (since tools only apply to development).

Component Store

External Component Defaults

The Component Store can now contain classes which provide default settings used when external components are added to a window or a report. These classes are:

– ?Remote Form Components
contains the default values for the OMNIS Web Client components.

– ?External Window Components
if present, contains the default values for external window components.

– ?External Report Components
if present, contains the default values for external report components.

To add new components to any of these classes, you must use the clipboard to copy from another library and paste to your class.

Wizard Support

The names of classes in the Component Store no longer need to conform to special conventions, as documented in the Using OMNIS Studio manual, with the exception of the field classes: these have fixed names that start with an underscore.

To replace the meaning of special names, there is now a new property of a class in the Component Store, $componenttype. This is has one of 5 values, represented by constants:

– kCompStoreHidden
  The class does not appear in the Component Store window

– kCompStoreNewClassDefault
  The class is the initial default for creating a new class of its type

– kCompStoreTemplate
  The class is a template from which a new class or classes can be created

– kCompStoreWizard
  The class is a wizard

– kCompStoreDesignObjects
  The class contains templates for design objects

The difference between a wizard and a template is that a wizard will prompt you for information required to create a class, whereas a template is simply copied to create the new class.

You can set $componenttype using the Notation Inspector, but is not visible using the class Browser/Property Manager, since it only applies to classes in the Component Store library. When the Component Store is displayed in the Browser, the class menu, and the class context menu, have one extra line at the end, which let you change the Component Store type of the class. When selected, it opens a small dialog, with radio buttons that enable you to select one of the types described above.

On the Class Browser, the class menu now has a new hierarchical menu, called **Wizards**, below the **New** menu. This has entries for each class type which contains wizards, templates or superclasses (via the $issupercomponent property), and each user class category (for example Net objects or DB2 extenders). When you select an item on the hierarchical menu, a dialog opens. Select the class, enter the name for the new class, and press OK or double click. The dialog creates a new class, and if it has a wizard, runs the wizard.

Note that as a result of this change, some of the class groups now have fewer items in them, e.g. the Net classes. The classes that were previously there are still available, but only by using those which are displayed.

The CompCopy utility has been updated to prompt you for the Component Store type of your own Component Store classes.

Icon Editor

The **File** menu in the Icon Editor has a new hierarchical menu **Open #ICONS**. There is an entry on this menu for each open library. If you select it, the editor opens #ICONS for that library.

The **Image** menu has a new item **Find Icon…**, which opens a small dialog in which you enter an icon id. It searches for the icon in the currently open icon file/table, and displays it.

SQL Browser

The SQL Browser now updates its session list dynamically, reflecting sessions opened and closed by other libraries.

Version Control System

Linked Classes Improvements

You can now link a class or component to more than one project in the same operation.

There is a new report to show which classes are linked to which projects.

The VCS component browser has a new icon to show classes which are linked. This is only shown in Detailed View and can be turned off from the VCS Options dialog.

Multiple Class actions

You can now delete or unlock multiple classes in a single operation.

Multiple Project actions

You can now delete or build multiple projects in a single operation.

Create new Project Components

You can now create new project components, from components in another project, using drag and drop.

New Reports

There are 2 new reports. The Associated Components report, which shows which classes are linked to which projects, and a report to show the privilege level of each user for each component.

Build Log

The build log now displays a total build time at the completion of the build process.

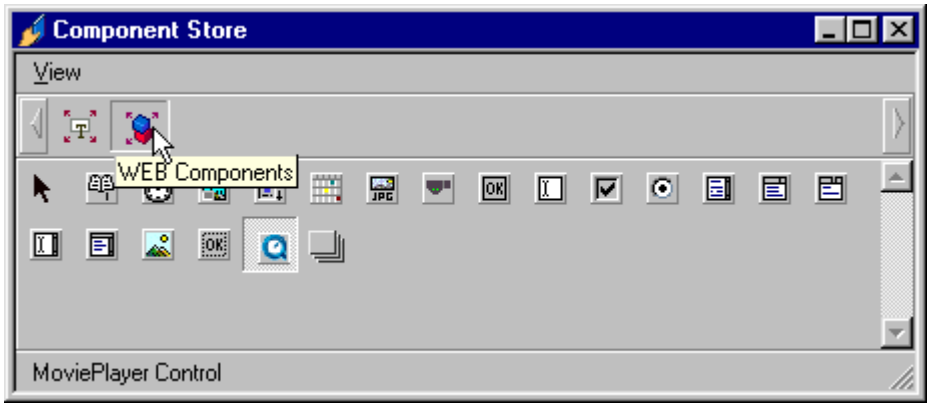
OMNIS Web Client

For OMNIS Studio 2.2 and 2.3 the OMNIS Web Client contained a number of enhancements, including new and improved components for web forms, QuickTime 4 support, and general communications improvements. In addition, the web client now provides automatic and dynamic download of new or out-of-date components. OMNIS Studio now contains the full release version of the web client for Netscape Navigator running under PowerMac and Linux.

New Web Components

OMNIS Studio 2.2 provides several new components for remote forms (web forms). You can use any of these components in your own remote forms, but any component used in a remote form must be available to the client. Using the new Automatic Component Download functionality you can deploy the basic field components, contained in the standard web client installer, and allow the client to download other components as required. See the next section in this chapter for further details about Automatic Component Download.

Note that all the new and updated web components are available and fully documented on the OMNIS website under a new section in the Component Gallery.



All the new web components are available in the **Component Store** under the **Web Components** button, and you can add them to your remote forms in exactly the same way as any other component in OMNIS; a remote form has to be open in design mode to see the components. Note all background and text web comps, such as the new Tile and Wash comps, are under the **Web Background Objects** button in the Component Store. Having added a web component to your remote form you can change its properties in the **Property Manager,** usually under the Custom tab, like any other object or component in OMNIS.

Jpeg

The **Jpeg** component is for displaying jpeg pictures in remote forms. You can specify an instance variable name in the **dataname** property of a Jpeg object.

Roll button

The **Roll button** is a new type of graphical pushbutton that highlights when you pass the mouse over it. It is available as a standard external component for window classes, and as a web component for remote forms. The new Welcome application uses the Roll button component extensively.

Roll button has a number of properties to control the look and behavior of the object when the mouse is placed over it. You can specify the image (**outsideimage** and **insideimage**) and text (**outsidetext** and **insidetext**) to be displayed when the mouse is either inside (over) or outside (not over) the roll button. You can also specify the text offset using the **textx** and **texty** properties, and set the spacing of multi-line text using **betweenlines**.

Roll button reports the **evIsInside** and **evClick** events, which have to be enabled in the **events** property for the object, as do all events for web objects.

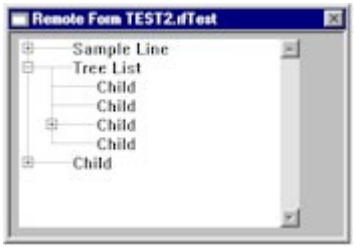
Button Area

The Button area component is an invisible area on the screen that reports an evClick or evDoubleClick event. Note you have to be enable these events in the **events** property for the object. You can use a button area over graphics or text, and in most respects it behaves like a standard button.

Timer

The Timer component triggers an evTimer event after a specified time and runs the associated event method; note you need to be enable the event in the **events** property for the object. You can specify a **timervalue**, which is interpreted using the setting of **useseconds** which should be kTrue for seconds (the default) or kFalse for milliseconds.

Tree



The **Tree** component presents a hierarchical list of values that you can expand and collapse by clicking on the nodes. The structure of the tree is determined by a list variable associated with the tree list and specified in its **dataname** property; note for remote forms the list must be an instance variable.

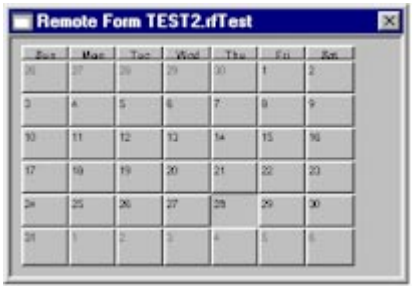
Several properties control the appearance of a tree list. When the **shownodeicons** property is set (kTrue), you can select the **defaultnodeicon** and the expand/collapse node in **expandcollapseicon**. Also you can position the node icons using **nodeiconspos** either on the node, on the left, or as set by the system.

The Tree component generates several events, providing they are enabled in the **events** property for the object. The events include:

- **evWTreeNodeClick**
  triggered when a node is clicked
- **evWTreeNodeDClick**
  triggered when a node is double-clicked
- **evWTreeNodeExpand**
  triggered when a node is expanded
- **evWTreeNodeCollapse**
  triggered when a node is collapsed

All these events can be detected in the $event() method for the tree list. All these events return the pNodeIdent parameter containing the ident of the node clicked or expanded. See the *OMNIS Programming* manual for details about tree list methods and in particular how to structure the list variable behind the tree list object.

Calendar



The **Calendar** component displays the current month in a grid and highlights today's date. It has many properties, which you can change in the Property Manager, that let you control the appearance of the calendar as a whole, and how today is displayed.

The Calendar component generates some events, providing they are enabled in the **events** property for the object. All the events can be detected in the $event() method for the object. The events include:

- **evDateChange**
  triggered when the date is changed
- **evMonthReset**
  triggered when the month is changed
- **evDateDClick**
  triggered when a date is double-clicked

Tile and Wash

The Tile and Wash components are under the **Web Background Components** button in the Component Store; note a remote form has to be open in design mode to see the components.

The **Tile** component requires an **iconid** of the icon or picture to be tiled. You can specify an id from the #ICONS system table in the current library, but make sure the page containing the object's icon is included in the **iconpages** property for the remote form.

The **Wash** component requires a **startcolor** and **endcolor** to specify the colors, and you can specify the wash **direction**.

Movie Player

The **MoviePlayer** component in OMNIS Studio 2.2 supports streaming video in the OMNIS Web Client, and allows the playback of Quicktime version 3 and 4 movies and sound files. The component has many properties which you can view and set in the Property Manager. If you are unsure what a property does, show the Help tips for the Property Manager (Right/Ctrl-click and select Help Tips option) and move your mouse over the property to display its description.

The **moviefile** property lets you specify the name and path of a movie or sound file. The **movieurl** property lets you specify the URL for a movie or sound file. Movies are streamed from the web server, thus removing the need for the file to be completely downloaded before viewing. Note you can only assign moviefile or movieurl, not both. Assigning one will automatically clear the other one.

Many of the other properties of the Movie Player component are self-explanatory. However the **action** property is required to allow you to control the movie or sound at runtime, since the object has no methods. The **action** property takes a constant, one of the following:

kQTActionNone	No action
kQTActionPlay	Plays the movie
kQTActionStop	Stops the movie
kQTActionPause	Pauses the movie
kQTActionReverse	Plays the movie in reverse
kQTActionStepFwd	Advances the movie by a single frame
kQTActionStepRev	Reverses the movie by a single frame
kQTActionGotoFront	Moves to the beginning of the movie
kQTActionGotoBack	Moves to the end of the movie

If the client does not have Quicktime 3 or 4 installed, then a Quicktime "Get4" logo displayed.

Automatic Component Download

The OMNIS Web Client now has the ability to download new or out-of-date components when a remote form is first accessed. OMNIS achieves this by checking the version of all components contained in a remote form against a database of components.

Component Database

The database of web components, called ctrlmgr.df1, is located in the Studio folder under the main OMNIS folder in the development and server versions. The database contains a record for each Web Client control for each platform. There is a new library ctrlmgr.lbs also in the Studio folder which manages this data file. You can access the Web Component Manager library using the Web Client option on the Tools add-ons menu. The component manager lets you add, delete, or extract components in the data file, although note that you can only add or extract on the platform to which the web component belongs. The components in the data file are compressed, using the new compress function.

When a client using the OMNIS Web Client connects to the OMNIS server, it returns a list of available components in the response to the connect request. The web client analyzes the form and the components already installed on the client, and if it needs to downloads any missing or out-of-date components. The web client prompts the user, asking them if they want to download the new or updated component. Assuming the user affirms, the Web Client downloads the component(s), and displays the remote form as usual.

Disabling Automatic Download

If you want to disable automatic component download, you can remove the ctrlmgr.df1 data file from the Studio folder.

Standard Web Client Installer

Note that the standard web client installer includes the standard form fields (formflds) and background components (formback) only. All other components are downloaded to the

client using the automatic download capability, unless you build your own installer containing some or all of the web components.

Compatibility

Old versions of the Web Client can safely be used against OMNIS Studio 2.2 servers which use ctrlmgr.df1.

New versions of the Web Client can safely be used against OMNIS Studio 2.1 servers, although of course no automatic download occurs.

Mac Web Client

OMNIS Studio 2.2 now contains a Netscape Navigator plug-in for MacOS and Linux. The Mac Web Client works with Netscape and MS Internet Explorer, although there's no secure comms support in MS Internet Explorer.

Remote Form Switching

In OMNIS Studio 2.2 you can switch from one remote form to another without destructing the form's task using the $ctask.$changeform(*new_form_name*) task method. For example, you could have an application containing a logon form and a payroll form. A user logs using the logon form then you can switch to the payroll form, without the initial connection and task being lost. Both forms would use the same remote task and task variables, and the payroll form would know what user is logged on.

Custom Cursors

Objects in remote forms now support both custom and built-in cursors. You can create your own cursor images and store them in the Userpic icon datafile, or the #ICONS system table in the current library.

All standard form fields, such as pushbuttons, button areas, now have the **cursor** property. You can enter the id of the image you want to appear as the cursor for the object. In addition, the **iconpages** property for remote forms now lets you include cursor pages in the icon pages that get sent to the client.

Pushbuttons with Icons

You can now assign a built-in or custom icon to a pushbutton in a remote form. You can create your own icons and store them in the Userpic icon datafile, or the #ICONS system table in the current library. To assign an icon to a pushbutton set its **iconid** property under the Appearance tab in the Property Manager. Remember to include the name of the icon page containing the button's icon in the **iconpages** property for the remote form.

Improved Error Handling

Certain errors returned by the Web Server previously resulted in potentially cryptic messages, such as "Inconsistent response length". Many of these Web Server generated errors actually come back from the Web Server as HTML. The OMNIS Web Client now writes this HTML to a file called omnis_error.htm, and the OK message generated to report the error contains the pathname of this file. Note that a subsequent error will overwrite the same file. This may be useful when you are setting up and debugging you web server.

Headed List boxes

Headed list boxes for remote forms have a number of enhancements, including several new properties.

Headed list boxes for remote forms have two new properties: **aligncolumns** and **alignheadings**, which specify the alignment of each column or the list box heading. You specify alignment as a string in the form XXX, where there is a character X for each column, and each character X can be one of L, R, or C, for Left, Right and Center alignment, respectively.

The column headings for a headed list box now respond to clicks. The **enableheader** property enables clicks for heading buttons. A click generates an **evHeaderClick** event, containing the column number in the pColumnNumber parameter.

The **boldheader** property controls whether the list box heading is shown bold or not.

The **disableresizecolumns** property disables column resizing.

Note headed list boxes now have a proper scroll range for their horizontal scroll bar.

Numeric edit fields

Numeric edit fields now use the client’s decimal point and thousands separator.

Serial Numbers

Up until now OMNIS web applications using the OMNIS Web Client were limited to 500 concurrent users. Now it’s possible to obtain a serial number that allows up to 10 times that number. The multi-user section of such a new serial number is in the form “nnnX” and means “ten times nnn” number of concurrent users can access the current OMNIS server.

Redirecting the client when using HTML forms

When using HTML forms (and not the web client), remote tasks can now return an URL in the form http://... or https://…, to redirect the client's browser. In OMNIS Studio 2.1, the omnisapi.dll/nph-omniscgi(.exe) always prefixed the return value from the remote task with http://<server name>. It now only prefixes the URL if it does not start with http:// or https://.

ATL based Active X

For OMNIS Studio 2.2 the OMNIS Web Client under Windows (ORFC.OCX) has been replaced with ORFC.DLL. This provides one or two benefits for the web client.

The old OCX was built using MFC, but is now built using Microsoft ATL (Active Template Library). This means that the web client on IE5 now scrolls correctly, and the $showurl() method supports frames. The size of the control is also reduced to around 100K.

ATL has one side effect in that the MFC version let you resize the plugin at runtime in the browser, but this is no longer supported. Therefore the HTML page must have the width and height set correctly. This was always the case with all other web client versions including the Windows Netscape plugin.

If you already deployed HTML pages for Win32 IE you may need to modify your pages to work with the new client.

Apache Server Extension

An Apache Module or server extension for Linux is now available. The module is called mod_omnis.so and can be found in the webclient/server folder in the main OMNIS tree. The Apache module works with the Red Hat 6.0 version of Apache, and other Linux distributions. This module is faster than the nph-omniscgi server extension.

To install the Apache module

- Copy mod_omnis.so to the Apache modules directory, e.g. on RedHat 6.0 this is /etc/httpd/modules

- Add the following lines at the end of access.conf:

```
<Location /omnis_apache>
SetHandler omnis-apache
</Location>
```

- Add the following lines to httpd.conf, at the end of each block of similar directives:

```
LoadModule omnis_module modules/mod_omnis.so
AddModule mod_omnis.cpp
```

You then need to restart httpd (restarting the system is the simplest way to do this). After installing the Apache module, you can address it using /omnis_apache in the WebServerScript parameter of the Web Client Netscape Plug-in or ActiveX.

API Open Source

Source for the Omnisapi.dll, Nph-omniscgi.exe, and the Apache module is now available on the OMNIS website via the Downloads area. This allows you to adapt the web

extension(s) for your chosen operating system, e.g., for a different version of Linux other than those currently supported by OMNIS Studio and the web client.

Pictures

True Color Shared Pictures

OMNIS Studio now supports true color (24 bit) shared pictures. These are implemented using the free source for PNG and ZLIB.

PNG or “Portable Network Graphics”, is a standard picture format, with a portable free source code implementation. ZLIB is a compression library, which also has a portable free source code implementation. You can find out more about PNG at the Web site for the World Wide Web Consortium, www.w3.org.

The $sharedpictures library preference now has three values, set using the constants:

– kSharedPicModeNone
  do not use shared pictures.

– kSharedPicMode256Color
  use the 256 color shared pictures from earlier releases.

– KSharedPicModeTrueColor
  use true color shared pictures.

If you use shared pictures, you are now recommended to use true color, since this will probably result in smaller stored images, and more realistic colors.

The data file browser now has a hierarchical menu from which you can choose no shared pictures, the old 256 color shared pictures, or true color shared pictures; this affects how pictures are converted when reorganising.

The window picture field and background picture object now have a $cachepicture preference, which defaults to kTrue. When kTrue, and a shared picture is being displayed, OMNIS keeps both the shared picture data, and a copy of the decompressed native OS picture - this uses more memory but results in faster drawing.

The reorganize data command has a new checkbox which only applies when the convert to shared option is checked. It indicates convert to true color shared pictures.

The Paste From File dialog now allows direct pasting of PNG files.

Note that conversion of images to true color loses color depth, unless you convert on a machine running in true color mode.

Picture Conversion

There are some new functions that let you convert picture data between a range of different formats. For OMNIS Studio 2.1 and later, these formats are:

- **CS8**
  OMNIS colour shared picture format (256 colours), including the internal OMNIS header.
- **CS24**
  OMNIS colour shared picture format (16 million colours) , including the internal OMNIS header.
- **CSC8**
  This is a 256 colour compressed format, which is an 8 bit PNG rather than 24 bit PNG format. Ideal for situations where images must have additional compression e.g. for downloading to the OMNIS Web Client. Obviously, this format may reduce the colour detail of an image that was originally 24 bit.
- **PNG**
  PNG format (Raw, as written on disk)
- **JPEG**
  JPEG format (Raw, as written on disk)
- **PCX**
  PCX format (Raw, as written on disk)

It is possible to add further formats using the external component interface.

The new functions are listed in a group called **Picture**, on the Functions tab of the Catalog. The functions are:

pictconvto()

pictconvto(*Character SrcFormat,Binary Src,Character DstFormat*)

Converts the supplied binary data (with or without our internal header) from the supplied source format to the supplied destination format. For example:

```
Do pictconvto("PCX",myPcxData,"JPEG") returns myJpegData
```

This converts the PCX data in myPcxData to JPEG.

pictconvfrom()

pictconvfrom(*Character SrcFormat,Binary Src*)

Converts from the raw data and the specified format to a picture value, which can be used in various OMNIS fields. For example, the following code lets you read a JPEG file from disk and display it:

```
ReadFile (“C:\MYFILE.JPG”) returns myJpegData
Do pictconvfrom(“JPEG”,myJpegData) returns myJpegData
Redraw {JPEG_CONTROL}
```

pictformat()

pictformat(*Binary Src*)

Returns a character string which contains the format of the picture data supplied. For example:

```
pictformat(myJpegData) will return “JPEG”.
```

Note that there are certain formats that pictformat cannot recognize (for example, OLE data, GIF), and in these cases, pictformat returns an empty string.

pictconvtypes()

Returns a single column list, which contains all the picture conversion types registered with OMNIS. For OMNIS Studio 2.1 and later, this contains the values “CS8”,”CS24”,”PNG”,”JPEG” and “PCX”.

Picture Conversion Example

In the following example, we convert the 24 bit colour shared images in the background picture objects of a remote form class, to the 8 bit PNG format CSC8.

```
Set reference fref to $root.$libs.THIN.$remoteforms.rfBooks.$bobjs
Set reference curBOBJ to fref.$first
While curBOBJ
   If curBOBJ.$objtype=kBackpicture
      If pictformat(curBOBJ.$picture)="CS24"
         ;  24 Bit picture
         Calculate picture as curBOBJ.$picture
         Calculate pict256 as pictconvto("CS24",picture,"CSC8")
         Calculate curBOBJ.$picture as pict256
      End If
   End If
   Set reference curBOBJ to fref.$next(curBOBJ)
End While
Save class {THIN.rfBooks}
```

Picture fields

The OMNIS picture field has a new property, $rawpictformat. This property defaults to <None>, in which case the picture field behaves as in previous versions. You can set it to the value of one of the picture formats used for picture conversion, such as "JPEG". It indicates the type of the data that can be stored in the picture variable associated with the picture field.

If you set this property, the **Paste from file** option converts the pasted file to the specified format.

Window Objects

Checkbox Lists

You can now disable individual lines in a checkbox list. The check box list has a new property, $statecolumn. It defaults to zero, which means all lines are enabled. $statecolumn must be >= zero. If it is non-zero, and less than or equal to the number of columns in the data list associated with the checkbox list, it identifies a column in the data list, which specifies the state of each line. This column must contain numeric values. A value of zero in this column means the line is disabled, and non-zero means the line is enabled. We recommend that you use the values zero and one, so that we can possibly use other values to provide additional feature(s) in future releases.

Masked Entry Fields

In previous versions, masked entry fields did not handle the localization of the decimal point and thousands separator characters.

To correct this, there are two new input mask characters:

'*' means the decimal point of a numeric value.

',' means the thousands separator of a numeric value.

If the input mask does not contain either of these two new characters, it will perform as in previous versions, to maintain compatibility. Note that in previous versions, the new characters were rejected as invalid, meaning that they do not introduce any compatibility issues.

If you use either or both of the new mask characters, OMNIS replaces each of them with the currently localized decimal point or thousands separator character, before displaying the mask.

It is recommended that you do not mix the asterisk and the decimal point character in the same mask, as results will be unpredictable.

Tree Lists

The tree list object now supports $addafter() and $addbefore(), for nodes.

The new runtime method $countall() recurses over the tree, and returns the total node count.

The new runtime method $getvisibleline(rItem) returns the line number of the supplied node in the currently displayed list of nodes, or zero if the node is not visible.

Tabbed Pane and Paged Pane Fields

Delete pane

There is a new context menu item on the window design mode context menu for a tabbed /paged pane field. The new item **Delete pane** is available if there are two or more tabs/pages, and if selected it will delete the current tab/page and all of the objects on that tab/page, after confirming via a yes/no dialog. Note that it will not delete objects marked as occurring on all panes.

Runtime Methods

$allpanes(Field Item Reference[,Boolean newval])

This method allows you to set or get the “all panes” Boolean attribute for the given field reference. If you supply newval the method sets “all panes” to newval. If you omit newval, the method returns the current value of “all panes”. For example:

```
;; If myField is shown on all tabs turn off all tabs.
If $cinst.$objs.TABPANE.$allpanes(myField)
Do $cinst.$objs.TABPANE.$allpanes(myField,kFalse)
EndIf
```

$panenumber(Field Item Reference[,Integer newval])

This method allows you to get or set the pane number which currently contains the specified field. If you supply newval, the method sets the pane number of the field to newval. If you omit newval, the method returns the pane number containing the field. For example:

```
;; If myField is shown on tab 2 move it to tab 3
If $cinst.$objs.TABPANE.$panenumber(myField)=2
Do $cinst.$objs.TABPANE.$panenumber(myField,3)
EndIf
```

Headed List Boxes

$columnheaderstyle(iColumnNumber[,summedStyleConstants]) is a new runtime method of the headed list box, which sets or gets the additional styles to apply to the column header of the specified column

It lets you specify some additional styles to use when drawing a column header. Additional means in addition to the field's text style (and bold if the bold header property is true).

Call it with just the column number to return the current value for the column.

$canresize… for Headed List Boxes and Grids

There are two new properties which determine if rows and columns of certain objects can be resized with the mouse at runtime. The $canresizeheader property of the headed list box has been renamed $canresizecolumns.

The complex grid, headed list box, data grid and string grid all have $canresizecolumns - this is True for a new object created via the component store, and for existing grids.

The complex grid also has the $canresizerows property.

For the complex grid, these properties do not affect sizing between the header and the main body of the grid, but they do affect sizing between the horizontal header or vertical header and the main body of the grid.

Data and String Grids

Data and string grids have a new runtime method, $enablecolumn(), which can disable or enable entry into a column which is not fixed (entry into fixed cells is always disabled).

– Grid.$enablecolumn(<column number>)
  returns kTrue if the column is enabled, or kFalse if the column is disabled.

– Grid.$enablecolumn(<column number>, kFalse)
  disables the column.

– Grid.$enablecolumn(<column number>, kTrue)
  enables the column.

Columns are initially enabled by default. Note that entry into a column is only enabled if the enter data state of the window allows, and both the grid field property $enabled is kTrue, and the column is enabled.

Mouse, key and status events

All window objects (except the report modify field) now have $mouseevents, $rmouseevents, $keyevents, and $statusevents properties. This provides finer control than in previous releases, where these properties are only available in the library preferences. An

object now gets the relevant event if either its own property or the library preference is true. These properties now appear on the Action tab of the Property Manager.

Window Status Bars

The window status bar now supports the $fieldstyle property.

External Components

Tray

There is a new non-visual object in the Tray library, which provides an interface to the tray in the Win32 task bar.

QuickTime 3

There is a QuickTime 3 component for the Win32 and Power Mac platforms.

Document Viewer

The Document Viewer is a new external component for viewing HTML documents in a window class. It will be extended to display RTF as well as HTML in OMNIS Studio version 3.0. The Document Viewer is used extensively in the new Studio 2.2 Welcome application, to display the OMNIS help and tutorial Html files.

When placed on a window you can change the properties of the HTML component under the Custom tab in the Property Manager. The component does not have a dataname property, rather you specify the path to an HTML text file in the **filename** property. You can set the filename in design mode or set it dynamically at runtime.

The HTML component has the following properties:

- **$filename**
  pathname to the HTML to be displayed by the component
- **$fontsizeadj** (runtime only)
  increases or decreases the size of the text in the current HTML file, in a range from -3 to 3 with the default being 0 (zero)
- **$eventhwnd** (runtime only)
  returns the hwnd reference of the HTML component when an event is triggered
- **$searchwords** (runtime only)
  contains a list of words that will be highlighted in the current HTML file; the words should be separated by spaces; the specified words and the background page surrounding the words are shown in their inverse

The HTML component has the following methods:

- **$startanimatescroll**(horzscrollunits,vertscrollunits,interval)
  scrolls the HTML component using the settings in horzscrollunits, vertscrollunits, interval

- **$stopanimatescroll**()
  stops scrolling the current HTML component
- **$pathtoapi**(path)
  converts an Html file name and path to a disk name and path; uses the correct separators for the correct operating system; performs the reverse of $pathtohtml()
- **$pathtohtml**(path)
  converts a standard disk name and path to a full Html file name and path; performs the reverse of $pathtoapi()
- **$getselectedtext**(text)
  returns any text currently selected in the HTML component

The HTML component reports many different events and you can write event handlers in the object's $event() method to handle them. When you create the HTML component from the Component Store a template $event() method is inserted to handle the basic events in the object at runtime. Note all the events for the HTML component return the ident of the field as the first event parameter.

The HTML component sends the following events:

- **evAnimateScrollEnd**
  sent when the HTML component is finished scrolling; returns the parameters pEventCode, pCtrlIdent
- **evEventTag**
  sent when an embedded custom Html tag is read; returns the parameters pEventCode, pCtrlIdent, pName, pValue
- **evExecTag**
  sent when an embedded custom Html tag is executed; returns the parameters pEventCode, pCtrlIdent, pTagName, pTagValues
- **evHyperlink**
  sent when a hyperlink is clicked; returns the parameters pEventCode, pCtrlIdent, pHRef, pName, pTarget, pTitle
- **evImagePluginCreate**
  sent when an embedded image plugin is invoked, such as an embedded Jpeg object; returns the parameters pEventCode, pCtrlIdent, pType, pProperties, pWindowRef, pWidthRef, pHeightRef
- **evPluginDestroy**
  sent when an embedded plugin is destroyed; returns the parameters pEventCode, pCtrlIdent
- **evSetTitle**
  sent when the Html document is read; returns the parameters pEventCode, pCtrlIdent, pTitle

– **evXCompPluginCreate**
sent when an embedded external component is invoked; returns the parameters pEventCode, pCtrlIdent, pComponentLib, pComponentCtrl, pProperties, pWindowRef, pWidthRef, pHeightRef

For example, the following $event() method for the Html component sets the title of the current window to the title of the Html file, and the method handles hyperlinks by switching to the specified Html file in the evHyperlink event.

```
On evSetTitle
   Do $cwind.$title.$assign(pTitle)
On evHyperlink
   Do $cobj.$filename.$assign(pHRef)
```

The following method traps the evHyperlink method and tests if the link is a link to an OMNIS library; if it is a library it uses the $pathtoapi() method to convert the link to a file path and opens the library, otherwise the method opens the specified Html document.

```
On evHyperlink
   If pos(".lbs",pHRef)
      Calculate lOmnisLibPath as pHRef
      Do $cobj.$pathtoapi(lOmnisLibPath)
      Open library (Do not close others) {[lOmnisLibPath]}
   Else
      Do $cobj.$filename.$assign(pHRef)
   End If
```

Class Caching

There is a new $root preference called $maxcachedclasses that controls how many classes OMNIS keeps cached in memory. After OMNIS reads a class from disk, it adds it to the class cache. If there are more than the number of classes specified in $maxcachedclasses in the cache, OMNIS removes one of the unmodified and unused classes from the cache, if there are any. Of course, this means that the next time the removed class is used, OMNIS has to load it from disk again.

$maxcachedclasses allows you to reduce the number of times classes are reloaded from disk, at a cost of greater memory usage, and some extra CPU time to manage the cache. From our own testing, we have observed an improvement in the performance of some applications (especially on the Mac platform), by increasing the cache size.

In OMNIS Studio 2.2 and earlier, $maxcachedclasses was fixed at 30. The default in 2.3 is 100. You can change the value, either using the notation, or using the Tools>>Options/Preferences menu item.

To allow you to tune your application, there are 2 new sys functions:

- sys(190) returns the number of times OMNIS has loaded a class from disk and added it to the cache.
- sys(191) returns the number of times OMNIS deleted a cache entry, when it added a class to the cache, meaning that a class may need to be reloaded.

Report Objects

HTML Raw Text Object

This is a new HTML report object, which inserts raw text into the HTML generated when printing a report to HTML. It has two properties:

– $::text
contains the calculation for the text.

– $addatend
set to kTrue if you require the text to be output at the end of the HTML table. If it is kFalse the text will be output in the individual cells within a table.

When the object outputs the value of $::text, it does not perform any character conversion or insert any HTML markers.

RTF Report Destination

This is a new custom device which provides the ability to generate reports in RTF format The RTF destination has four parameters, which you can access using the report destination dialog. The options on the dialog include:

– **Destination:**
The output filename.

– **Ignore Images:**
Images are only supported in Word95 onwards, so if you are using old RTF readers (for example write.exe on windows) you may want to check this option to produce smaller image-free documents.

– **Link Images:**
Supported only in Word 95 onwards. This option places links in the document to JPEG images, reducing the size of the RTF document file, but resulting in a document which is not self-contained. If you do not check this option, the resulting document contains embedded images.

– **Convert multi-lines to single line:**
Multi-line is difficult in RTF, as Word supports text boxes but other readers do not. If you check this option, multi-line text will appear in a single line and wrap at the end of the page. If you do not check this option, the RTF device simulates a text box, by placing a carriage return at the end of each line, and setting the tab position of the next

line so that it falls below the previous one. If users wish to reformat their document after OMNIS has produced it, this may not be the best option.

Note: The device does not perform any line drawing or background coloring, since these are only possible in Word RTF format and not the standard RTF format.

External Components

You can now use runtime notation on external component report instance objects. For example, this allows you to modify the properties of a graph on a report, on a per-record basis.

External components now use true color format when adding images to reports.

Page Count

There is a new page count external component, which allows you to include text such as “Page M of N” in your reports. Obviously, when you use this component, output of the report to the spooler is delayed until the report has been completely generated, since the total page count is not known until then.

Runtime Environment

$singleinstance

There is a new Boolean notation property, $prefs.$singleinstance. This property is only applicable to the Win32 versions of OMNIS Studio, where it defaults to kFalse. If you set it to kTrue, OMNIS will only let you start a single copy of OMNIS Studio from the folder where OMNIS Studio is installed. This means that if you have 2 installations of OMNIS Studio on your machine, they both have a $singleinstance property; one installation might only allow a single instance, and the other might allow multiple instances.

If the user starts OMNIS Studio for which $singleinstance is kTrue, and the instance is already running, the initial instance of OMNIS Studio will be restored (if minimized) and brought to the front. In this case, if the user opens OMNIS Studio by selecting a library (e.g. double clicking on a library in Explorer), after coming to the front, the single instance opens the library. It does not close any libraries which are already open.

If you set $singleinstance to kTrue, and more than one instance of OMNIS Studio is already open, these instances will be unaffected. The property does not apply until you try to start another copy.

$singleinstance is stored in the registry entries for the particular OMNIS Studio installation.

The Windows Registry

From OMNIS Studio 2.1, each installation of OMNIS Studio has its own registry entries. In earlier versions, the key contained the major version number, but multiple installations of the same major version shared the registry entries.

Startup Library Load Order

At startup, OMNIS now loads libraries in the startup folder, in case-insensitive alphabetical order of their pathname. This provides you with the ability to control the order in which the libraries start, by renaming library files.

Replacement Edit Menus

You can now provide a replacement edit menu, with lines that provide standard operations such as copy and paste, and where these standard lines enable and disable according to the current selection.

There is a new command, Standard menu command, which allows you to select a standard command such as copy or paste, to execute when the user selects a menu line.

As a background, in the 1.x and 2.0 versions of OMNIS Studio, if you set the $event method for a menu line to:

```
Call method OBSOLETE COMMAND *Edit/11002
```

and no other commands apart from comments, selecting the menu line causes OMNIS to execute a standard OMNIS edit menu cut command.

The new command "Standard menu command" behaves in exactly the same way, and simply allows you to use standard menu commands without using an obsolete command.

If you replace the standard edit menu, each line in the replacement menu which uses Standard menu command (or the old Call method technique) now enables correctly for the following commands: Undo, Cut, Copy, Paste, Clear, Select all, Paste from file, Insert Object and Links.

DDE

Windows platforms only. In previous versions, DDE only worked with the most recently opened non-private library. In OMNIS Studio 2.1 and later, there is a new notation item, $root.$modes.$cddelib. This is the internal name of the library OMNIS uses for DDE. If you do not assign this item, DDE behaves as in earlier versions. If you assign this item, OMNIS uses the specified library for DDE, rather than the most recently opened non-private library. When assigning $cddelib, the library must be one which is accessible to the current task. You can set $cddelib to an empty string, to revert to the original functionality.

Typically, you would use this in the startup task of your library, for example:

```
Calculate $root.$modes.$cddelib as $clib().$name
```

When you close the library, OMNIS clears $cddelib. Note that you cannot assign this item using the Notation Inspector.

Changing Separator Characters

In OMNIS Studio 2.2 and later, the $root.$prefs has a new method $separators() for getting and setting the separators used in decimal numbers, function syntax, and import/export.

Method	Description
$separators()	When called with no arguments, $separators() returns the current separator values into a 5 character string. The characters in this string are: Character 1: the decimal point character Character 2: the decimal number thousands separator Character 3: the function parameter separator Character 4: the import/export decimal point character Character 5: the import/export comma delimiter character When called with a 5 character string, it sets the separator characters from the values in the string.

Graphs

Non-Visual Graph Object

The graphs component now provides a non-visual object, which can generate a graph and store it in a picture variable. One key application of the non-visual object is generating graphs on-the-fly, to return to a browser running the OMNIS Web Client.

To use this object, add an Object variable to an OMNIS class, and set the subtype to 'Graph'. The object provides the ability to manage graph properties and methods, and draw the graph into a picture variable, without needing a field on a window or report.

The non-visual graph object has many of the methods and properties of its visual counterpart; use the interface manager for more details. The key method is $snapshot(width, height), which returns a picture value with the specified width and height.

For example:

```
Calculate graphObj.$dataname as MyList
Calculate graphObj.$maintitle as "My Graph"
Calculate graphObj.$majortype as kGRlines
Calculate graphPict as graphObj.$snapshot(320,200)
```

The $getobject method may be used from the graph object to manipulate the grouped properties.

MySQL DAM

An ODBC DAM for Linux which supports MySql was provided in OMNIS Studio version 2.3. See the *OMNIS Programming* manual for details about logging onto server DBMSs.

ODBC Access Control

ODBC Access Control is a new feature of the OMNIS data file ODBC driver. It is the ability to restrict ODBC access to particular tables and fields, to particular users.

This provides significantly greater control than the current public data property implemented in OMNIS Studio 2.0 and OMNIS 7^3 Version 7.0.

This feature is backwards compatible with the public data property.

Table Access Control

In Studio 2.0, the public data property is stored as a single byte in the data dictionary information, which has the value zero for no access, or one for public access.

In Studio 2.1 and later, there is a new 32 bit field at the end of the data dictionary information for a file. The single byte used by the current public data property becomes unused. We call the new 32 bit field the table access mask. Each bit in the mask corresponds to a user group, and if a bit is set to true, the table belongs to that user group. The mask can have any value, meaning that a table can belong to any subset of the 32 user groups (note that OMNIS 7^3 version 7.1 will only support 31 groups).

For example, you might decide to have user groups for Sales, Accounts, R&D and Marketing. In other words, you assign 4 specific bits of the table access mask, to these groups. A table can then belong to zero or more of these groups.

You can set the value of the table access mask for a file class, in a similar way to setting the public data property in version 2.0.

You can store simple ODBC user profiles in a data file. The profiles are stored as a table, ODBCUSERS, with three columns: ODBCNAME, ODBCPASSWORD and ODBCACCESSMASK. You can administer this table using the ODBC Admin add-on tool.

An ODBC connection to an OMNIS data file has a user access mask, which identifies the user group or groups to which the user belongs. A table is accessible on that connection if the result of and'ing the user access mask, with the table access mask, is non-zero. Or, in other words, the table is accessible to the user if the table and the user belong to at least one common group. Inaccessible tables cannot be used in ODBC queries, and are not visible to ODBC data dictionary queries.

How do we set the user access mask? When a user logs on to a data file using ODBC, the driver performs some additional checks. If there is not a table called ODBCUSERS, the driver assigns a user access mask of 1 to the ODBC connection. This corresponds to the current public data value, and allows for backwards compatibility. If there is a table called ODBCUSERS, the driver requires <username> and <password> parameters in order to connect. These are case insensitive, and are OMNIS driver specific ODBC keywords, meaning that the driver will prompt for them (if allowed by the ODBC API), if they are not in the configuration information for the data source. The driver looks up the user in the ODBCUSERS table. If the user is not present, the connection cannot be established; otherwise, the driver sets the user access mask for the connection to the value stored in the ODBCUSERS row for the user.

Note that if users store their passwords in data source definitions, the passwords will be stored in an encoded form. However, another user could use the data source definition with the encoded password to log on. Therefore, the best security occurs when users do not store their passwords in data source definitions.

Field Access Control

An additional level of control is provided by field access control. The driver applies this after identifying the tables which are accessible to the user.

Each field has a field access mask. Again, this is a 32 bit mask. For backwards compatibility, its initial value has all bits set to true, thereby enabling access to all fields in accessible tables.

You can modify the field access mask using the file class editor.

The field access mask works just like the table access mask. A user can access a field if it belongs to an accessible table (according to the definition above), and if the field and the user belong to at least one common user group. Inaccessible fields cannot be used in ODBC queries, and do not appear in ODBC data dictionary queries.

User Interface

The file class properties include the ODBC access mask, which can be modified via a checkbox list.

The file class editor has a control which drops down a checkbox list from which you can set the field access mask.

The data file browser displays the value of the table access mask in the slot listing. After changing table or field access masks, you may need to update the data dictionary in the data file using the data file browser or the update data dictionary command.

Printing a file class or the slot listing now includes the access mask values.

OMNIS SQL

You can execute the following SQL with OMNIS SQL to set ODBC access masks:

```
execute odbc.setaccess(Towns.Postcode,3)
```

sets the mask of column Town.Postcode to 3 (the numeric value corresponding to the mask group bits set). Similarly

```
execute odbc.setaccess(Towns,3)
```

sets the mask for table Towns. Also,

```
execute odbc.getaccess(Towns) or
   execute odbc.getaccess(Towns.Postcode)
```

gets the access mask. You can retrieve the result of a successful getaccess using sys(139). Note that sys(139) returns a negative value if bit 32 of the mask is set, since this is the sign bit of a 32 bit signed integer.

This only works when OMNISSQL is connected directly to a data file specified by the host name.

Notes

- A user can have an empty password.
- User names can comprise the characters 0-9, A-Z, a-z, _ and extended Ansi characters >= 128.
- Passwords cannot contain the character ~.
- The ODBCUSERS table must have a table access mask with all bits set to true.
- The new ODBC Driver will be compatible with data files previously accessed by the current OMNIS Studio 2.0/OMNIS 7^3 version 7.0 ODBC driver (we call this the old driver from now on). The old driver will NOT be able to access data files containing the new mask information - all tables in such a data file will become non-public, since when OMNIS writes the new 32 bit table access mask to the data dictionary, it clears the public data byte.

SQL Server 7.0

Support for MSSQLServer7.0 has been added to the ODBC DAM on all platforms. The Microsoft Windows 32 bit drivers support the new additions to MSSQLServer7.0 such as

varchar and char data lengths up to 8000 characters. Other drivers provided by vendors such as Intersolv and Visigenic still place an upper limit of 255 for these types and data retrieved exceeding this is truncated. This also occurs with the Microsoft 16 bit driver. The latest Macintosh versions of the Intersolv and Visigenic SQL Server drivers do not return date values correctly. The use of the Intersolv driver on 32 bit platforms can cause an EXCEPTION_ACCESS_VIOLATION error to occur on the server when inserting into columns of type ntext. Also the same error is generated from the Intersolv Macintosh driver when manipulating the new Unicode types nchar and nvarchar. Use of these new types should be avoided for these drivers.

Port Profiles

OMNIS Studio 2.0 and earlier versions let you configure ports using either the Set port parameters method command, or the report destination dialog. The configuration options available there map on to only a subset of the options possible using the operating system port configuration API calls. As a result, there have been some problems related to port configuration. The fact that Win16, Win32, and the Mac, all have different port configuration data structures, further compounds the problems.

A **port profile** is a named collection of information sufficient to completely describe the operating system configuration of a port.

Before going into detail, here is some clarification about a few points:

All Macintoshes have 2 ports available, the printer port and the modem port. These ports are both serial ports, and OMNIS stores separate configuration data for each port.

Windows machines can have a number of ports, some of which are serial ports (e.g. COM1:), and others which are parallel ports (e.g. LPT1:). OMNIS only remembers the configuration data for a single port.

Therefore, on the Windows platforms, the port profile information corresponds to the information required to set the fields in a DCB for a serial port (Win16's DCB is different to Win32's). There is no port profile for a parallel port, since it requires no operating system configuration information.

On the Mac platforms, the port profile information corresponds to the information required to make the SerReset and SerHShake API calls.

The information required to completely configure a port therefore comprises:

1. The port profile (not required for parallel ports)

2. Characters per inch.

3. Lines per inch.

The "Page sizes" entered on the corresponding tab of the report destination dialog for the port destination remain unchanged.

Port Profile Management

Each port profile is stored in a file. The sub-directory PORTS of the OMNIS folder contains the port profiles.

The port profile file contains:

– An indicator of the platform to which the profile corresponds - Win16, Win32 or Mac. This allows for runtime sanity checking.

– The name of the profile, to be used in the report destination dialog, and as an argument to the Set port parameters method command. We store this in the file, rather than use the file name as the profile name, to allow for readable names on all platforms. Note that profile names are not case sensitive.

– The profile data.

Note that this means that each profile file only contains the data for a single platform.

You can create and edit profiles using the port profile editor. The profile editor must run on the platform for which the profiles are to be created. For Win32 and Power Mac development versions, the port profile editor is an add-on tool. If you wish to edit port profiles using a runtime version, there is a folder in the root of the Studio CD, called RTTools, containing the editor library, which you copy to your hard disk, and open to run the editor.

Port Profiles at Runtime

Report Destination Dialog

When the report destination is port, the parameters tab of the dialog displays the port configuration information. There is now a drop down list on this tab, which contains a list of profile names, and one additional entry, "Use options below". When the latter is selected, the port works in the same way as OMNIS Studio 2.0 and earlier. When a profile is selected, OMNIS configures the port using the information in the profile. Note that the drop down list and configuration fields on this tab are disabled if you select a parallel port.

Set port parameters command

You can now use

```
Set port parameters {Profile name}
```

When the command executes, it first checks the entire parameter string against the list of profiles. If it matches an entry in the list, the command uses the profile to configure the port; otherwise, the command treats the parameter string as a parameter list, and behaves like OMNIS Studio 2.0 and earlier.

Notation

The relevant $port… notation such as $portparity is unassignable when a port profile is selected.

There is one new assignable notation item, $portprofile, which defaults to empty for backwards compatibility. This is the port profile for the current port on the Mac, or just the single port profile on Windows.

The Euro Character

To support the Euro character, the character mapping table has been modified, which provides cross-platform mapping of characters in libraries and data files, between the Mac and Ansi character sets.

The Euro is code position 0x80 in the Ansi character set.
The Euro is code position 0xDB in the Mac character set.

The character mapping table used in previous releases maps characters as follows:

Ansi 0x80 maps to Mac 0xA5
Ansi 0xAF maps to Mac 0xDB

We have therefore swapped the mappings to that:
Ansi 0x80 maps to Mac 0xDB (the correct Euro mapping)
Ansi 0xAF maps to Mac 0xA5

This means that the Ansi "overscore" character (like an underscore, but at the top of the character cell) now maps to a Mac blob.

Note that the Euro character is supported by Mac OS 8.5, and by some Win32 operating systems. For details of Win32 support, go to www.microsoft.com – you may need to download an update.

String Tables

You can now set up a string table which enables you to have multiple copies of the same string. Typically, you would have a different copy of the string for a different language or country. You can then select the current language or country at runtime.

The add-on tools include the string table editor.

There are functions listed under StringTable in the catalog, which let you change the current language/country, and read strings for the current language/country.

There is a StringLabel background object, which enables you to reference a string in the string table from a label on a window or a report.

Each string is identified by an id, and each language/country is identified by a column in the string table.

Web Externals

The WEB externals have been replaced with a new implementation. This is essentially compatible with the externals from previous releases, but see the updated documentation in the **Reference** section of this document for details. In particular, there is a new approach to handling MIME content, available on Win32 and the Power Mac only.

SQL Classes

If you define a list from a SQL class, the columns in the list have a new property $excludefromwhere. This works in a similar way to the existing properties $excludefromupdate and $excludefrominsert. It defaults to kFalse, but if you set it to kTrue, OMNIS excludes the column from where clauses it generates. OMNIS only inspects this property for columns which it normally includes in the where clause (often only a few, if you are using primary keys).

Errata

The following errata section contains corrections to the printed and Adobe Acrobat documentation.

Version Control System

Build Completion Method

This feature was actually present in Studio 2.0, but was omitted from the documentation. You can specify a library that the VCS will open (and as a consequence run its startup task), after a build has completed.

To specify the library, use the code

Do $root.$modes.$dotoolmethod(kEnvToolVcs,'$x_vcsBuildOption',['pathName'])

where pathName is the full path name to the library. You can pass an empty string to clear this setting.

Building Locked Classes

This feature was actually present in Studio 2.0, but was omitted from the documentation.

You can indicate which classes will be locked in the built library i.e. which classes cannot be developed any further or viewed in the IDE.

Graphs

The graphs PDF file (graphs.pdf) refers to $group. This should actually be $::group.

$serialise

The $serialise() method has been previously documented as $serialize.

Reference

The following information supplements that provided in the *OMNIS Reference* manual.

Functions

sys()

These changes to the sys() function apply in OMNIS Studio 2.2 and later.

sys(6)
returns 'U' if running on Linux/Unix.

sys(8)
returns "UNIX" if running on Linux/Unix.

sys(9)
returns the path separator for the platform as follows:

\ (back slash)	for Windows
/ (forward slash)	for Linux/Unix
: (colon)	for Mac

sys(116)
returns '1' if files names are all lowercase.

encstr()

encstr(*string*[,*key*]) encodes the *string* using the *key*. If omitted, OMNIS uses a default value for the *key*.

The return value of encstr is a string that is difficult to decode without knowing the key. OMNIS uses encstr to encode passwords stored in the ODBCUSERS table, so that it is impossible to use a hex editor to obtain a password from a data file (see the ODBC Access Control section in this document).

decstr()

decstr(*string*[,*key*]) decodes the *string* using the *key*. If omitted, OMNIS uses a default value for the *key*.

Note that decstr(encstr(string,key),key) = string.

compress()

Compress is a new function in OMNIS Studio 2.2 and later.

compress(*binary*) compresses the specified *binary* variable and returns the binary compressed data. It uses the ZLIB compression algorithm to compress the binary variable.

uncompress()

Uncompress is a new function in OMNIS Studio 2.2 and later.

uncompress(*binary,uncompressed length*) uncompresses the specified *binary* variable (containing the result of a previous call to compress) and returns the binary uncompressed data. Note you need to pass in the length of the original uncompressed data.

Commands

Data Files

The *Open data file* and *Prompt for data file* commands have a new check box option "No conversion by runtime". When checked, the runtime version of Studio will refuse to convert the selected data file, if it needs converting.

Web Commands

This section describes the new Web commands which are supplied with OMNIS Studio 2.1 and later. These commands replace the old Web Enabler commands which previously shipped with the OMNIS 7 and Studio products. The new command set is essentially backwards compatible, and in most cases, code written using the old commands is source code and binary compatible with the new commands. The major differences are:

- Various commands have new optional parameters. If such a parameter is omitted, the behavior is that of the old command set.
- There are many new error codes. Not all of the old error codes are generated.
- FTPGetLastStatus has been renamed FTPSetConfig.
- WebDevError has been renamed WebDevSetConfig
- There is one new command: HTTPSetProxyServer.
- ALL commands now return an error status, which is ALWAYS less than zero if an error has occurred. Moreover, ALL command errors (with the exception of the “call would block” error) invoke the WebDevError method – see the command WebDevSetConfig for details.

– ALL commands which accept a port, also accept a service name. On Windows, this is resolved to a port number using the services file on the local system; on the Macintosh, this is resolved using a hard-coded lookup table.

– You can now have blocking sockets on the Macintosh platforms. You can cause a blocked WEB command to abort, using ctrl-break on Windows platforms, and command-period on Macintosh platforms. For compatibility, all commands which return a new socket, with the exception of TCPAccept (for example: TCPConnect, HTTPGet), return a blocking socket on Windows, and a non-blocking socket on the Macintosh. Use TCPBlock to change its state.

– The Content Manager commands (CM…) have not been replaced in the new command set. There are new mechanisms provided by the MailSplit and Pop3Recv commands, to handle MIME content (these are available on the Win32 and Power Macintosh platforms only). You can still use the old CM… commands if you wish, but bear in mind that the current version of these commands is the last that OMNIS Software will ship.

– Similarly, the binfile commands (ReadBinFile and WriteBinFile) have not been replaced. The current Fileops commands provide adequate functionality, and you should use these instead.

You can tell if your system has the new command set by looking in the external folder of the OMNIS installation. If there is a single WEB external, named wecommnd, rather than separate externals for FTP, HTTP etc., you have the new command set.

CGIDecode

Reversible: NO **Flag affected:** NO

Parameters: Stream[,MapPlusToSpace {Default kTrue}]

Returns: *DecodedStream*

Syntax: CGIDecode(*Stream[,MapPlusToSpace]*)

You use CGIDecode to turn CGI-encoded text back into its original form. It is the inverse of CGIEncode.

When a client uses HTTP to invoke a script on a WEB server, it uses the CGI encoded format to pass the arguments to the server. This avoids any ambiguity between the characters in the argument names and values, and the characters used to delimit URLs, and the argument names and values.

stream is an OMNIS Character or Binary field containing the information to decode.

MapPlusToSpace is a Boolean value. When kTrue, in addition to performing a standard CGI decode operation, the command maps all instances of the '+' character in the input stream, to the space character.

DecodedStream is an OMNIS Character or Binary field that receives the resulting CGI-decoded representation of the *stream* argument.

Note: The HTTPHeader, HTTPParse and HTTPPost commands automatically perform CGI encoding or decoding, as appropriate.

CGIEncode

Reversible: NO **Flag affected:** NO

Parameters: stream
Returns: *EncodedStream*

Syntax: CGIEncode(*stream*)

When a client uses HTTP to invoke a script on a WEB server, it uses the CGI encoded format to pass the arguments to the server. This avoids any ambiguity between the characters in the argument names and values, and the characters used to delimit URLs, and the argument names and values.

You use CGIEncode to map text into the CGI encoded format.

Stream is an OMNIS Character or Binary field containing the information to encode.

EncodedStream is an OMNIS Character or Binary field that receives the resulting CGI-encoded representation of the *stream* argument.

Note: The HTTPHeader, HTTPParse and HTTPPost commands automatically perform CGI encoding or decoding, as appropriate.

FTPChmod

Reversible: NO **Flag affected:** NO

Parameters: Socket, Filename, Mode
Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPChmod(*Socket*, *Filename*, *Mode*

FTPChmod changes the protection mode of a remote file on the connected FTP server.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

Filename is an OMNIS Character field containing the pathname of the remote file.

Mode is an OMNIS Character field containing the system-dependent file-protection specifier to apply to the named file. Many FTP servers accept the Unix-style Owner/Group/World 3-digit Read/Write/Execute scheme (for example, 754 = Owner Read/Write/Execute, Group Read/Execute World Read-Only). Consult the documentation for the remote system to determine the acceptable syntax for this argument.

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPConnect

Reversible: NO **Flag affected:** NO

Parameters: ServerAddr, Username, Password [,Port]

Returns: *Socket* (< 0 if an error occurs)

Syntax: FTPConnect(*ServerAddr,Username,Password[,Port]*) Returns *Socket*

FTPConnect establishes a connection to the specified FTP server.

ServerAddr is an OMNIS Character field containing the hostname or IP address of the FTP server.

Username is an OMNIS Character field containing the user ID with which the command will log on to the server.

Password is an OMNIS Character field containing the password for the user ID.

Port is an optional number or service name, which identifies the TCP/IP port of the FTP server. If you omit this parameter, it defaults to 21, the standard FTP port. If you use a service name, the lookup for the service will occur locally.

Socket is an OMNIS Long Integer field, which receives the result of the command. If the command successfully establishes a connection and logs on to the server, *Socket* has a value >= 0; you pass this value to the other FTP commands, to execute requests on this connection.

FTPCwd

Reversible: NO **Flag affected:** NO

Parameters: Socket, NewDir

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPCwd(*Socket,NewDir*)

FTPCwd changes the working directory for the specified FTP connection.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

NewDir is an OMNIS Character field containing the new working directory. The contents of this string are system-dependent. FTPCwd accepts anything for this argument, but the remote FTP server may not. Most FTP servers accept Unix-style path and file specifications with path and file separated by slashes, such as

/drive/user/subdirectory/filename.extension

Most FTP servers accept the Unix conventions for abbreviations for special directory specifications, that is, ".." for the next higher sub-directory, and "~userid" for the home directory of a particular user ID.

Some FTP servers also accept system-specific directory path formats, that is, Macintosh colon-separated as in Macintosh HD:My Folder:My File or VMS-style path and file specifications, as in SOME$DISK:[USER.SUBDIRECTORY]FILENAME.EXTENSION;1.

Consult the documentation for the server to determine the authoritative acceptable directory path specifications. When in doubt, try the Unix style.

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPDelete

Reversible: NO **Flag affected:** NO

Parameters: Socket, Filename

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPDelete(*Socket*, *Filename*) Returns *Status*

FTPDelete deletes a file on the connected FTP server.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

Filename is an OMNIS Character field containing the pathname of the remote file to delete.

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPDisconnect

Reversible: NO **Flag affected:** NO

Parameters: Socket

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPDisconnect(*Socket)*

FTPDisconnect closes a connection to an FTP server.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPGet

Reversible: NO **Flag affected:** NO

Parameters: Socket, RemoteFile, LocalFile [,FileType, Creator]

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPGet(*Socket,RemoteFile,LocalFile[,FileType, Creator]*)

FTPGet downloads a file from an FTP server. The file is transferred using the currently specified transfer type of ASCII or binary, as specified by the FTPType command. It is important that you set the transfer type correctly for each file you download, since an incorrect transfer type will result in a bad downloaded copy of the file.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

RemoteFile is an OMNIS Character field containing the pathname of the remote file to download.

Note: The remote filename may not be acceptable to the local system.

LocalFile is an OMNIS Character field containing the pathname of the downloaded file. If the file already exists, FTPGet will overwrite it with the downloaded file.

FileType and *Creator* are optional arguments, which the command uses on the Macintosh platforms only. These specify a file type and creator for the downloaded copy of the file. If you omit these arguments when calling FTPGet on a Macintosh, they default as follows:

- For ASCII transfer type: *FileType* = TEXT, *Creator* = ttxt
- For binary transfer type: *FileType* = TEXT, *Creator* = mdos

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPGetBinary

Reversible: NO **Flag affected:** NO

Parameters: Socket, RemoteFile, BinField

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPGetBinary(*Socket,RemoteFile,BinField*)

FTPGetBinary downloads a file from an FTP server into an OMNIS binary variable. The file is transferred using binary transfer mode.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

RemoteFile is an OMNIS Character field containing the pathname of the remote file to download.

BinField is an OMNIS Binary or Character field that will receive the contents of the remote file.

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPGetLastStatus

Reversible: NO **Flag affected:** NO

Parameters: [ServerReplyText]

Returns: *Status*

Syntax: FTPGetLastStatus(*[ServerReplyText]*)

FTPGetLastStatus returns status information corresponding to the last FTP command executed.

ServerReplyText is an optional OMNIS Character field parameter, into which FTPGetLastStatus places the dialog that occurred on the connection to the FTP server, for the last FTP command executed. For example, if you execute FTPPwd, and then call FTPGetLastStatus, ServerReplyText might contain:

-> PWD

<- 257 “/” is current directory.

Note that “->” prefixes text sent to the server, and “<-“ prefixes text received from the server.

Status is an OMNIS Long Integer field which receives the return status of the last FTP command executed. This information is really redundant, but is provided for compatibility. The value returned is one of the negative error codes.

FTPList

Reversible: NO **Flag affected:** NO

Parameters: Socket, List [, Pathname, Mode]

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPList(*Socket,List*[,*Pathname,Mode*])

FTPList lists files on the FTP server.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

List is an OMNIS List field containing a single column of type Character. This list receives the file listing information, one line per file, returned by the remote FTP server. The list is dependent on the type of the remote server and may be in long or short format, depending on the *Mode* parameter.

Note: Very often, FTP servers return long-format listings in a Unix file listing format. At a minimum, this file information contains the filename, but usually includes other information. The OMNIS method must parse this information to find the filename and other information. For example

ListItem					
total 123					
drwxr-xr-x	4	userid	mygroup	Jan 1 1999	.
drwxr-xr-x	6	root	root	Jan 1 1999	..
-rw-------	1	userid	mygroup	Jan 16 1998	myfile
-rw-r—r—	2	userid	mygroup	Jan 16 1998	myotherfile

Where the columns in the character string correspond to protection, file size, username and group of the file owner, the date last modified and the name of the file. The files "." and ".." represent the current and parent directories, respectively, which may neither be retrieved nor changed.

The file information may not be neatly spaced into columns as in this example. Columns are separated with one or more spacing characters (space, tab, and so on).

Pathname is an optional OMNIS Character field that contains a pathname or wildcard specification for the files to include in the listing. If omitted, the default is to list all of the files in the current directory on the FTP server.

Mode is an optional numeric value which indicates whether the server should return a short or long format listing. If omitted, it defaults to zero.

Code	Meaning
0	Filename-only listing
1	Long-format listing

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPMkdir

Reversible: NO **Flag affected:** NO

Parameters: Socket, DirName

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPMkdir(*Socket,DirName*)

FTPMkdir creates a new directory on the FTP server.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

DirName is an OMNIS Character field containing the pathname of the new directory to create on the server.

Note: The name of the new directory must follow the convention and file-naming rules of the remote system. Not all users will have permissions to create new directories on arbitrary directories on the remote system. Default file-access permissions apply to the new directory.

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPPut

Reversible: NO **Flag affected:** NO

Parameters: Socket, LocalFile, RemoteFile

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPPut(*Socket,LocalFile,RemoteFile*)

FTPPut uploads a local file to the FTP server. The file is transferred according to the currently specified transfer type of ASCII or binary as specified by the FTPType command. It is important that you set the transfer type correctly for each file you upload, since an incorrect transfer type will result in a bad uploaded copy of the file.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

LocalFile is an OMNIS Character field containing the pathname of the file to upload.

RemoteFile is an OMNIS Character field containing the pathname of the destination file on the FTP server.

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPPutBinary

Reversible: NO **Flag affected:** NO

Parameters: Socket, BinField, RemoteFile

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPPutBinary(*Socket*,*BinField*,*RemoteFile*)

FTPPutBinary uploads the contents of an OMNIS binary variable to a remote file on the FTP server. The data is transferred using binary transfer mode.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

BinField is an OMNIS Binary or Character field containing the data to transfer.

RemoteFile is an OMNIS Character field containing the pathname of the destination file on the FTP server.

Status is an OMNIS Long Integer field which receives the result of executing the command.

FTPPwd

Reversible: NO **Flag affected:** NO

Parameters: Socket

Returns: *ServerDirectory* (pathname if no error, or a number < 0)

Syntax: FTPPwd(*Socket*)

FTPPwd gets the pathname of the current directory on the FTP server.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

ServerDirectory is an OMNIS Character field that receives the pathname of the current directory. If this is a number less than zero, an error occurred.

Note: The value returned depends upon the operating system of the remote server. Many FTP servers return a Unix-style pathname, but do not assume that this is the case.

FTPReceiveCommandReplyLine

Reversible: NO **Flag affected:** NO

Parameters Socket

Returns: *Reply* or a number < 0 if an error occurs

Syntax: FTPReceiveCommandReplyLine(*Socket*)

FTPReceiveCommandReplyLine returns the next line of the reply following an FTPSendCommand. You have to determine if the reply is multi-line, and if so issue further receive commands to get the remainder of the reply. FTPReceiveCommandReplyLine will timeout after 60 seconds if it does not receive a reply.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

Reply is an OMNIS Character variable containing the reply from the server.

For example:

```
FTPSendCommand(lvSocket,'pwd') Returns #1
FTPReceiveCommandReplyLine(lvSocket) Returns lvReply
; might return the string
257 "/vol1/ftp/omnis/" is current directory
```

FTPRename

Reversible: NO **Flag affected:** NO

Parameters: Socket, OldName, NewName

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPRename(*Socket,OldName,NewName*

FTPRename renames a remote file.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

OldName is an OMNIS Character field containing the pathname of the file to rename.

NewName is an OMNIS Character field containing the new pathname for the file

Status is an OMNIS Long Integer field which receives the result of executing the command.

Note: Local filename conventions may not be acceptable to the remote system.

FTPSendCommand

Reversible: NO **Flag affected:** NO

Parameters Socket, Command

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPSendCommand(*Socket,Command*)

FTPSendCommand sends a command to the FTP server.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

Command is an OMNIS Character variable containing the command and its parameters.

Status is an OMNIS Long Integer field which receives the result of executing the command.

For example:

```
FTPSendCommand(lvSocket,'pwd') Returns #1
FTPReceiveCommandReply(lvSocket) Returns lvReply
; might return the string
257 "/vol1/ftp/omnis/" is current directory
```

FTPSetConfig

Reversible: NO **Flag affected:** NO

Parameters: Proc [,ActiveOnly {Default zero for no;1 for yes},68kPort1, 68kPort2]

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPSetConfig(*Proc[,68kPort1, 68kPort2])*

FTPSetConfig provides the FTP commands with configuration information.

Proc is an OMNIS Character field containing the name of an OMNIS method used to report the progress of FTP operations which transfer data (FTPGet, FTPGetBinary, FTPList, FTPPut and FTPPutBinary); for example MYLIBRARY.MYCODE/MYPROC. You can clear the current setting for the FTP progress proc, by passing an empty value.

ActiveOnly is an optional parameter. A value of 1 causes all FTP to be active, rather than the default, which is use passive FTP if the server supports it. Normally, you would not select ActiveOnly FTP; this is provided as a possible work-around for servers with which passive FTP is causing problems. You can find a fuller explanation below of passive and active FTP.

68kPort1 and *68kPort2* are optional parameters, which are only used on the 68k Macintosh platform. If specified, they indicate a range of TCP/IP ports which the FTP data transfer

commands use cyclically to perform active FTP data transfer. You can find a fuller explanation below of passive and active FTP.

Status receives the result of executing this command.

FTP data transfer commands call the progress proc (if specified) while data transfer is in progress. This allows you to indicate progress to the user. The commands call the progress proc with three parameters:

- Socket: the FTP socket on which the operation is occurring
- TransferredSoFar: the number of characters transferred so far, or for FTPList, the number of lines received so far.
- TotalToTransfer: the total number of characters that need to be transferred; note that this is only available when executing FTPPut or FTPPutBinary.

The 68kPort parameters are needed on the Macintosh 68k platform, because MacTCP does not dynamically allocate port numbers when passively opening a connection. In practice, you will probably not need to set these values, because by default, the FTP data transfer commands always first attempt to use passive mode to transfer data. In passive mode, the client initiates a data connection to the server. This is the recommended mode of operation (see RFC1579, "Firewall Friendly FTP). Most FTP servers support passive mode, although there are some which do not. In this case, if the attempt to use passive mode fails, the FTP commands use active mode to transfer data. In this case, the server initiates the data connection to a port on the client, and we therefore need a range of port numbers which the commands can use for this purpose. The parameters *68kPort1* and *68kPort2* specify this range of port numbers.

FTPSite

Reversible: NO **Flag affected:** NO

Parameters Socket, Parameters

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPSite(*Socket*,*Parameters*) Returns *Status*

FTPSite issues a host specific SITE command to the FTP server.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

Parameters is an OMNIS Character variable containing the host specific command and its parameters.

Status is an OMNIS Long Integer field which receives the result of executing the command.

For example:

```
FTPSite(lvSocketNum,"FILETYPE=JES") Returns lvStatus
; issues the FTP command SITE FILETYPE=JES
```

FTPType

Reversible: NO **Flag affected:** NO

Parameters: Socket, FileType

Returns: *Status* (0 if no error, < 0 if an error occurs)

Syntax: FTPType(*Socket*,*FileType*)

FTPType specifies the type of data transfer used by FTPGet and FTPPut, as ASCII or binary. In ASCII mode, line separators and other text formatting characters will be changed to the characters required by the local or remote system. In binary mode, line separators and other text formatting characters are not changed. If the information to be transferred is not text, use FTPType to change the transfer mode to binary. Otherwise, binary files such as archives, images, OMNIS Libraries, and executable files may be corrupted by the processing of bytes that coincide with text-formatting characters.

Socket is an OMNIS Long Integer field containing a socket opened to an FTP server using FTPConnect.

FileType is a number indicating the type of subsequent FTPGet and FTPPut transfers on this socket.

Value	Transfer Mode
kFalse/Zero	ASCII
kTrue/One	Binary

Status is an OMNIS Long Integer field which receives the result of executing the command.

HTTPClose

See TCPClose – HTTPClose has identical behavior.

HTTPGet

Reversible: NO **Flag affected:** NO

Parameters: Hostname, URI[, CGIList, HeaderList, Service|Port]

Returns: *Socket* (< 0 if an error occurs)

Syntax: HTTPGet(*Hostname,URI* [,*CGIList, HeaderList, Service*|*Port*])

HTTPGet is a client command that submits a GET HTTP request to a Web server.

Hostname is a Character field containing the hostname or IP address of the Web server.

URI is a Character field containing the URI to GET from the Web Server. For example, "/default.html", or "/cgi-bin/mycgiscript"

CGIList is an optional parameter. It is an OMNIS list with two character columns. The list contains the CGI arguments to be appended to the URI. There is one row for each CGI argument. For example

Attribute	Value
Name	John Smith
City	Podunk
Alive	On
Submit	Please

Note: Before the values are sent to the Web server, HTTPGet automatically performs any CGI encoding required to pass special characters in the arguments. There is no need to call the CGIEncode command.

HeaderList is an optional parameter. It is an OMNIS list with two character columns.. The list contains additional headers to add to the headers of the HTTP GET request. Note that the header name excludes the ':', which HTTPGet inserts automatically when it formats the header.

For example

Header name	Value
User-Agent	My Client
Content-type	text/html

Service|*Port* is an optional parameter that specifies the service name or port number of the server. If you specify a service name, the lookup for the port number occurs locally. If you omit this argument, it defaults to the port number specified in the hostname, or if none is present, it defaults to 80, the default port for HTTP.

Socket receives the result of the request. HTTPGet opens a connection to the Web server, and formats and sends an HTTP GET request to the server. If the command succeeds, it returns the socket number for the connection to the WEB server; otherwise, it returns an

error number which is less than zero. After successfully issuing HTTPGet, you should call HTTPRead to read the response from the server; ALWAYS call HTTPClose to close the connection and free the socket.

HTTPGet adds the following header fields by default:

Attribute	Value
Accept	*/*
User-Agent	OMNIS Software – OMNIS

Note: After calling HTTPGet, you can call HTTPSend to send your own content, before you read the response, provided that you include Content-type and Content-length headers in the *HeaderList*.

HTTPHeader

Reversible: NO **Flag affected:** NO

Parameters: Socket, Status, HeaderList

Returns: *Length* (or a value < 0 if an error occurs)

Syntax: HTTPHeader(*Socket,Status,HeaderList*)

HTTPHeader is a server command that sends an HTTP standard header to an HTTP client, for example, an OMNIS application or a Web browser. HTTP headers are normally hidden from Web clients, but convey very useful information regarding the status and contents of the Web page. An OMNIS method must send a header back to a connected Web browser in order to have its results properly displayed.

Socket is an OMNIS Long Integer field containing the number of a socket that has already been opened for a TCP/IP client, usually a Web browser or OMNIS application that requires and can understand HTTP.

Status is an OMNIS Long Integer field containing an HTTP status code. The status code may change the way in which any following HTML or other information displays on the Web browser. The following table contains the status codes which HTTPHeader recognises. Other status codes are accepted, but HTTPHeader then sends “Unknown status” as the text for the code.

Code	Meaning
200	The request was completed successfully
201	The request was a POST method and was completed successfully. Data was sent to the server, and a new resource was created as a result of the request.
202	A GET method returned only partial results.
204	The request was completed successfully, but there is no new information. The browser will continue to display the document from which the request originated.
301	The requested URL has moved permanently
302	The requested URL has moved temporarily
304	The GET request included a header with an If-Modified-Since field. However, the server found that the data requested had not been modified since the date in this field. The document was not resent (the Web browser will probably display it from cache).
400	The request syntax was wrong
401	The request requires an Authorization field but the client did not specify one. Usually results in a username and password to be displayed
403	Access is forbidden
404	The request URL could not be found.
500	The server has encountered an internal error and cannot continue with the request.
501	The server does not support this method
502	Bad gateway
503	Service unavailable

HeaderList is an OMNIS list with two character columns. The list contains the headers to send. Note that HTTPHeader automatically sends some headers, so do not provide those (see below).

At a minimum, for OMNIS to return normal Web-page HTML text to the client, you should send a header containing the line:

Header name	Value
Content-type	text/html

HTTPHeader automatically includes the following lines in all HTTP response headers:

Attribute	Value
Content-type	text/html (only if the *HeaderList* does not contain a Content-type header)
Date	The current GMT date and time in HTTP header format
Server	OMNIS
MIME-version	1.0

Length is an OMNIS Long Integer field which receives the number of characters sent, or an error code less than zero.

HTTPOpen

Reversible: NO **Flag affected:** NO

Parameters: Hostname[, Service|Port]

Returns: *Socket* (< 0 if an error occurs)

Syntax: HTTPOpen(*Hostname*[,Service|*Port*])

HTTPOpen is a client command that opens an HTTP connection to a Web server.

Hostname is a Character field containing the hostname or IP address of an HTTP server. For example:

www.myhost.com or 255.255.255.254

Service|*Port* is an optional parameter that specifies the service name or port number of the server. If you specify a service name, the lookup for the port number occurs locally. If you omit this argument, it defaults to 80, the default port for HTTP.

If HTTPOpen succeeds, s*ocket* receives a positive number which is the socket for the new connection to the server. Otherwise, *socket* receives a negative error code.

HTTPPage

Reversible: NO **Flag affected:** NO

Parameters: URL[, Service|Port]

Returns: *Page*

Syntax: HTTPPage(*URL*[,Service|*Port*])

A client command that retrieves the content of the Web page specified by the URL, into an OMNIS Character or Binary variable.

Note: HTTPPage allows you to get HTML text source through a server, transparently and without additional coding.

URL is an OMNIS Character field containing a standard Web page URL of the form
http://domaininfo.xxx/path/webpagepage

Service|Port is an optional parameter that specifies the service name or port number of the server. If you specify a service name, the lookup for the port number occurs locally. If you omit this argument, it defaults to the port number specified in the URL, or if none is present, 80, the default port for HTTP.

The primary role of HTTPPage is to grab, simply and quickly, the HTML text source of the page specified by the URL. The URL may also specify a CGI name and arguments, but it is simpler to access CGIs by using the HTTPPost or HTTPGet functions.

If an error occurs, the command returns a negative number to *Page*. Otherwise, *Page* receives the contents of the specified URL. In other words, it receives the complete HTTP response for the URL, including the status line and the headers.

HTTPParse

Reversible: NO **Flag affected:** NO

Parameters: Message, HeaderList, Method, HTTPVersion[, URI, CGIList]

Returns: *Status* (Zero for success, < 0 if an error occurs)

Syntax: HTTPParse(*Message,HeaderList,Method,HTTPVersion[,URI,CGIList]*)

HTTPParse is a server command to parse HTTP header information from an incoming request message.

Message is an OMNIS Character or Binary field containing the full text of an HTTP request message.

HeaderList is an OMNIS list with two character columns. The list receives the headers extracted from the request message, one line per header.

For example, after the call, the list might contain entries such as:

Attribute	Value
Date	The current GMT date and time in HTTP header
User-Agent	NCSA Mosaic for the X Window System/2.4 libwww/2.12 modified
Accept	/
Content-type	application/x-www-form-urlencoded
Content-length	1234

Note: HTTPParse automatically strips the colons after the attribute names.

Method is an OMNIS character field that receives the type of HTTP method being requested, for example, GET, POST, or HEAD.

HTTPVersion is an OMNIS Character field containing the version of HTTP. For example, 1.0.

URI is an OMNIS Character field that receives the name of the URI to be processed. At a minimum, the URI is a single slash, so every URI returned from HTTPParse is of the form /URLName.

Note: Due to the presence of the leading slash, a simple OMNIS equality string comparison to the name of the URI fails. Use the pos() function or similar parsing mechanism to find the URI name. The trailing question mark of a GET-method CGI, which separates the URI path from the CGI arguments, is stripped by HTTPParse.

CGIList is an OMNIS list field with two character columns. It receives the CGI arguments present in the request, either extracted from the URL, or extracted from content of type "application/x-www-form-urlencoded". For example, if the following HTML form is the submitted from a browser:

Name:
City:
Are you alive?

and the user types in *John Smith, Podunk* and checks the City field, after HTTPParse, *CGIList* contains:

Attribute	Value
Name	John Smith
City	Podunk
Alive	Yes
Submit	Please

Note: Before the data is stored in the list, HTTPParse automatically decodes any CGI encoding required to pass special characters. There is no need to call the CGIDecode command.

HTTPPost

Reversible: NO **Flag affected:** NO

Parameters: Hostname, URI [,CGIList, HeaderList, Service|Port]

Returns: *Socket* (< 0 if an error occurs)

Syntax: HTTPPost(*Hostname,URI* [,*CGIList, HeaderList, Service*|*Port*])

HTTPPost is a client command that submits a POST HTTP request to a Web server.

Hostname is a Character field containing the hostname or IP address of the Web server.

URI is a Character field containing the URI to GET from the Web Server. For example, "/default.html", or "/cgi-bin/mycgiscript"

CGIList is an optional parameter. It is an OMNIS list with two character columns. The list contains the CGI arguments to be posted to the URI. These will be sent as content of type

“application/x-www-form-urlencoded”. There is one row for each CGI argument. For example

Attribute	Value
Name	John Smith
City	Podunk
Alive	On
Submit	Please

Note: Before the values are sent to the Web server, HTTPPost automatically performs any CGI encoding required to pass special characters in the arguments. There is no need to call the CGIEncode command.

HeaderList is an optional parameter. It is an OMNIS list with two character columns.. The list contains additional headers to add to the headers of the HTTP POST request. Note that the header name excludes the ‘:’, which HTTPPost inserts automatically when it formats the header.

For example

Header name	Value
User-Agent	My Client
Content-type	text/html

Note that because CGI arguments are sent as content, you can only supply your own Content-type and Content-length headers if you do not supply CGI arguments.

Service|*Port* is an optional parameter that specifies the service name or port number of the server. If you specify a service name, the lookup for the port number occurs locally. If you omit this argument, it defaults to the port number specified in the hostname, or if none is present, it defaults to 80, the default port for HTTP.

Socket receives the result of the request. HTTPPost opens a connection to the Web server, and formats and sends an HTTP POST request to the server. If the command succeeds, it returns the socket number for the connection to the WEB server; otherwise, it returns an error number which is less than zero. After successfully issuing HTTPPost, you should call HTTPRead to read the response from the server; ALWAYS call HTTPClose to close the connection and free the socket.

HTTPPost adds the following header fields by default:

Attribute	Value
Accept	*/*
Content-length	The length of the content (Only if you supply CGI arguments)
Content-type	application/x-www-form-urlencoded (Only if you supply CGI arguments)
User-Agent	OMNIS Software – OMNIS

Note: After calling HTTPPost, you can call HTTPSend to send your own content, before you read the response, provided that you include Content-type and Content-length headers in the *HeaderList*.

HTTPRead

Reversible: NO **Flag affected:** NO

Parameters: Socket, Buffer [,Type]

Returns: *ReceivedCharacterCount* (< 0 if an error occurs)

Syntax: HTTPRead(*Socket,Message[,Type]*)

HTTPRead is a client and server command that reads a complete HTTP request message or response. Servers use it to read requests, and clients use it to read responses.

Socket is an OMNIS Long Integer field containing the socket number of an open HTTP connection.

Buffer is an OMNIS Character or Binary field into which HTTPRead places the received request or response.

Type is an optional parameter. It is a Boolean value, where zero indicates server behavior, and non-zero indicates client behavior. If omitted, it defaults to zero.

ReceivedCharacterCount is an OMNIS Long Integer field which receives the number of characters placed in *Buffer*. If an error occurs, an error code less than zero is returned here.

Note: HTTPRead always operates in blocking mode, and will timeout after a minute of inactivity on the connection. The server reads until the HTTP request header is complete, and it has received content of the correct size. The client behaves similarly, but will also treat graceful closure of the connection as marking the end of the response.

HTTPSend

See TCPSend – HTTPSend has identical behavior.

HTTPServer

Reversible: NO **Flag affected:** NO

Parameters: WebProc[, Port]

Returns: *Status* (Zero for success, or < 0 if an error occurs)

Syntax: HTTPServer(*WebProc[,Port]*)

HTTPServer invokes a listening socket on a specified port, to receive incoming HTTP requests. This function shows an OMNIS working message with the count of accepted connections. HTTPServer calls a user-specified OMNIS method each time a new connection arrives. The user function receives the socket number for the new HTTP connection.

WebProc is an OMNIS Character field containing the name of the OMNIS method to be called when a connection arrives. The method receives one parameter, the number of the socket for the new HTTP connection. For example, MYLIBRARY.MYCODE/MYPROC.

You may read and write to the parameter socket with HTTPRead, HTTPSend, or HTTPHeader commands or a TCP equivalent (TCPSend; for example).

Port is an OMNIS Integer field that is optionally used to indicate the port number on which HTTPServer listens for connections. If omitted, the port number defaults to 80.

Caution: You must close the socket with HTTPClose before quitting the OMNIS method.

The command returns an integer *status*, which is less than zero if an error occurs.

Stopping HTTPServer

Once started, HTTPServer runs indefinitely until it is stopped. There are two ways to stop HTTPServer:

1. Press the Cancel button on the working message displayed by the command.
2. Set the OMNIS flag to false before returning from the WebProc method. Obviously, you need to make sure the flag is true before returning, if you wish to process further connections

HTTPSetProxyServer

Reversible: NO **Flag affected:** NO

Parameters: [Hostname,Service|Port]

Returns: *Status* (Zero for success, or < 0 if an error occurs)

Syntax: HTTPSetProxyServer(*[Hostname,Service|Port]*)

HTTPSetProxyServer is a client command which sets the HTTP proxy server to be used for the client commands HTTPGet, HTTPPage, and HTTPPost. After specifying a proxy server in this way, these commands connect to the proxy server rather than the Web server indicated by their parameters. They then use a different HTTP syntax to request Web pages via the proxy server.

Hostname is the hostname or IP address of the proxy server.

Service|*Port* is an optional parameter that specifies the service name or port number of the proxy server. If you specify a service name, the lookup for the port number occurs locally. If you omit this argument, it defaults to 8080.

To clear the proxy server setting, and subsequently connect directly to Web servers, call this command with no (or empty) parameters.

The command returns an integer *status*, which is less than zero if an error occurs.

HTTPSplitHTML

Reversible: NO **Flag affected:** NO

Parameters: Message, TagtextList

Returns *Status* (Zero for success, or < 0 if an error occurs)

Syntax: HTTPSplitHTML(*Message*,*TagtextList*)

HTTPSplitHTML is a client function to parse the HTML from a Web page into an OMNIS list. The HTML tags are parsed out of the text, so that it easier to write a program that grabs the Web page content or interprets the tags from a form.

Message is an OMNIS Character or Binary field containing the text of the content portion of a Web page, including HTML tags.

TagtextList is an OMNIS list defined to have three columns, all character. Column 1 contains the opening HTML tag, column 2 the actual page text, and column 3 the closing HTML tag.

The command returns an integer *status*, which is less than zero if an error occurs.

HTTPSplitURL

Reversible: NO **Flag affected:** NO

Parameters: URL, Hostname, URI

Returns: *Status* (Zero for success, or < 0 if an error occurs)

Syntax: HTTPSplitURL(*URL*,*Hostname*,*URI*)

HTTPSplitURL is a server and client function which splits a full URL into a hostname and a path (that is, a URI). Useful for following HREF links on pages.

URL is an OMNIS Character field containing a standard Web page URL of the form http://host.mydomain.com/path/webpage.html

Hostname is an OMNIS character field that receives the hostname parsed out of the URL argument. For example, given the URL, above, the hostname portion would be host.mydomain.com

URI is an OMNIS Character field that receives URI parsed out of the URL. For example, given the URL, above, the URI would be /path/webpage.html.

The command returns an integer *status*, which is less than zero if an error occurs.

MAILSplit

Reversible: NO **Flag affected:** NO

Parameters: Message, HeaderList, Body

Returns: *Status* (Zero for success, or < 0 if an error occurs)

Syntax: MAILSplit(*Message,HeaderList,Body*)

MAILSplit parses an Internet e-mail message. On the Win32 and Power Macintosh platforms, it can also decompose MIME content.

Message is an OMNIS Character or Binary field containing the complete text of an Internet e-mail message, including the header. Messages in this form are returned in the MailList argument of the POP3Recv command. For example

```
Received: by omnis-software.com with SMTP; 12 Aug 1996 11:49:59 -0700
Received: (from someone@localhost) by netcom8.netcom.com (8.6.13/Netcom)
id LAA09789; Mon, 12 Aug 1996 11:46:45 -0700
Date: Mon, 12 Aug 1996 11:46:45 -0700
From: someone@somedomain.com (PersonalName here)
Message-Id: <199608121846.LAA09789@netcom8.netcom.com>
To: someoneelse@somedomain.com
Subject: This is an e-mail subject


Hello from OMNIS Software, Inc.
```

HeaderList is an OMNIS list with two character columns. The list receives the information from the e-mail message header as attribute/value pairs. There is one row for each item in the header. For example, assuming the e-mail message above:

Attribute	Value
Received	by omnis-software.com with SMTP; 12 Aug 1996 11:49:59 -0700
Received	(from someone@localhost) by netcom8.netcom.com (8.6.13/Netcom) id LAA09789; Mon, 12 Aug 1996 11:46:45 –0700
Date	Mon, 12 Aug 1996 11:46:45 –0700
From	someone@somedomain.com (PersonalName here)
Message-Id	<199608121846.LAA09789@netcom8.netcom.com>
To	**Error! Reference source not found.**
Subject	This is an e-mail subject

Note: Two header lines may have the same attribute name. This is within the RFC822 message header specification. In this case, the HeaderList has two lines with the same Attribute name, as with Received in the above example. Long header lines that are split and continued in the message header are concatenated into one line in the list, as with the second Received attribute in the above example. The colon at the end of the attribute is stripped.

The *Body* parameter can be either an OMNIS character field or an OMNIS list.

If *Body* is an OMNIS character field, MailSplit returns the body of the e-mail message into this variable, minus the header. In the example: Hello from OMNIS Software, Inc. Note, however, that if the body contains MIME content, the HeaderList only receives headers up to and excluding the MIME-Version header, and the body receives the rest of the message, starting with MIME-Version. This allows you to use the old Content Manager CM… commands to analyse the MIME content. This mechanism is no longer recommended, but is maintained for compatibility.

On the Win32 and Power Macintosh platforms, you can pass an OMNIS list as the *Body* parameter. In this case, the *HeaderList* receives all of the headers, and the *Body* list receives either a single line containing the message body (if the message does not have MIME content), or a line for each MIME body part in the message body (if the message has MIME content). We discuss how MIME content is handled in this way below.

The command returns an integer *status*, which is less than zero if an error occurs.

MIME Content

MIME content can be thought of as a tree, which has a single root node, the message. Each node in the tree has a MIME type and a MIME subtype.

Non-leaf nodes have the type “multipart”, and these contain other nodes, which themselves can be multipart. A non-leaf node does not contain data.

Leaf nodes have other types, such as “text” and “application”, and these contain data. The type “message” can also be considered a container, but the MailSplit (and SMTPSend) commands treat messages as leaf nodes. If you wish to decompose a message contained in MIME content, you need to call MailSplit again for that message.

Each node in the tree is referred to as a body part.

The *Body* list receives a representation of the MIME content tree, with a line for each body part. Before calling MailSplit, define a list with eight columns:

Column	Contains
Level	A long integer which indicates the level of this node in the tree. The single root node has level zero. The next level down is one, and so on. This will become clearer in some examples below.
Content-type	The type of this body part e.g. “text” or “multipart”
Content-subtype	The sub-type of this body part e.g. “plain”
Filename	The name of the file corresponding to this body-part. Used for leaf-nodes which are file attachments.
Character data	If the content-type is “text” or “message”, this column contains the data. Leaf nodes only.
Binary data	If the content-type is not “text”, “message” or “multipart”, this column contains the data. Leaf nodes only.
Character-set	The character set of the data. The commands only understand us-ascii and iso-8859-1. The latter is equivalent to the Ansi character set used on the Windows platforms. Character data in any other character set will not be handled correctly.
Content-Transfer-Encoding	How the data is encoded: “base64”, “quoted-printable”, “7bit” etc. The command handles decoding from base64 and quoted-printable, meaning that the data in the character and binary columns above has been decoded. On the Power Macintosh, character data in the iso-8859-1 character set has been converted to the Macintosh character set. On both Win32 and the Power Macintosh, the command replaces CRLFs with the OMNIS newline character.

Some example lists:

A message sent by a mailer such as Outlook Express, containing both text and HTML versions of the message text:

Lev	Content-type	Content-subtype	File	Char	Bin	Char-set	Encoding
0	multipart	alternative					
1	text	plain		From Bob		iso-8859-1	quoted-printable
1	text	html		<!DOCTYPE HTML…		iso-8859-1	quoted-printable

A message sent by a mailer such as Outlook Express, containing both text and HTML versions of the message text, and having a single file attachment:

Lev	Content-type	Content-subtype	File	Char	Bin	Char-set	Encoding
0	multipart	mixed					
1	multipart	alternative					
2	text	plain		From Bob		iso-8859-1	quoted-printable
2	text	html		<!DOCTYPE HTML…		iso-8859-1	quoted-printable
1	application	octet-stream	app.h		This is my file data …		base64

POP3Recv

Reversible: NO **Flag affected:** NO

Parameters: Server, User, Pass, List[, Delete, StsProc, MaxMessages]

Returns: *Status* (Zero for success, or < 0 if an error occurs)

Syntax: POP3Recv(*Server*,*User*,*Pass*,*MailList*[,*Delete*,*Status*,*MaxMessages*])

POP3Recv retrieves Internet e-mail messages from a POP3 server into an OMNIS list. If an error occurs, the command returns a value less than zero to *Status*; in this case, all mail may not have been received. The socket opened to the POP3 server is always closed before the command returns.

Server is an OMNIS Character field containing the IP address or hostname of a POP3 (Post Office Protocol Level 3) server that will serve e-mail to the client running OMNIS. Examples: pop3.mydomain.com or 255.255.255.254.

User is an OMNIS Character field containing the account that receives the mail on the designated server. Usually an account user name, for example, Webmaster.

Pass is an OMNIS Character field containing the password for the account specified in the *User* parameter, for example, Secret.

List is an OMNIS list field defined to contain a single column of type character. The column receives the Internet e-mail messages, one per line. The column variable should be large enough to receive the e-mail message, including the header. When using OMNIS 7, the list should be defined with the store long data option selected.

Delete is an OMNIS Boolean field which, if true, indicates that the messages will be deleted from the server once they have been downloaded into MailList. The default is false, so messages remain on the server if the argument is omitted.

StsProc is an optional parameter containing the name of an OMNIS method that POP3Recv calls with mail receive status messages. The method can display a status message while the POP3 process proceeds. POP3Recv calls the method with no parameters, and the status information in the variable #S1.

MaxMessages is an optional parameter which specifies the maximum number of messages to be downloaded by this call to POP3Recv. If omitted, all available messages are downloaded.

POP3Stat

Reversible: NO **Flag affected:** NO

Parameters: Server, User, Pass [,StsProc]

Returns: *WaitingMessages* (or an error code < 0)

Syntax: POP3Stat(*Server,Username,Password[,StsProc]*)

The POP3Stat command retrieves the number of Internet e-mail messages waiting for a particular user on a specified POP3 server. If an error occurs, the command returns a value less than zero to *WaitingMessages*. The socket opened to the POP3 server is always closed before the command returns.

Server is an OMNIS Character field containing the IP address or hostname of a POP3 server that will serve e-mail to the client running OMNIS. For example: pop3.mydomain.com or 255.255.255.254.

User is an OMNIS Character field containing the account that receives the mail on the designated server (usually an account user name, for example, Webmaster).

Pass is an OMNIS character field containing the password for the account specified in the *User* parameter, for example, Secret.

StsProc is an optional parameter containing the name of an OMNIS method that POP3Stat calls with status messages. The method can display a status message while the POP3 process proceeds. POP3Stat calls the method with no parameters, and the status information in the variable #S1.

WaitingMessages is an OMNIS Long Integer field which receives an error status, or the number of e-mail messages waiting to be collected on the specified server for the specified account.

SMTPSend

Reversible: NO **Flag affected:** NO

Parameters: Server, From, To, Subject, Body[, CC, BCC, Name, StsProc, Pri, XtraHdrs]

Returns: *Status* (Zero for success, or < 0 if an error occurs)

Syntax: SMTPSend(*Server,From,To,Subject,Body*
[,*CC,BCC,Name,StsProc,Pri,XtraHdrs*])

SMTPSend sends an Internet e-mail message via an SMTP server. It returns a *Status* value less than zero if an error occurs.

Server is an OMNIS Character field containing the IP address or hostname of an SMTP server that will accept e-mail requests from the client running OMNIS, for example, smtp.mydomain.com or 255.255.255.254.

From is an OMNIS Character field containing the RFC 822 Internet e-mail address that will be placed in the header to identify the sender. Recipients can reply to this address, for example, webmaster@www.omnis-software.com.

To is either an OMNIS Character field or an OMNIS list field. If the field is character, it contains the RFC 822 Internet e-mail address to which the e-mail will be sent, for example, webmaster@www.omnis-software.com. If the field is a list, it has a single character column, which contains one RFC 822 Internet e-mail address per row.

Subject is an OMNIS character field containing the subject of the e-mail message.

Body is

- either an OMNIS Character or Binary field containing the body of the e-mail message; the text appears as the actual e-mail message
- or (Win32 and Power Macintosh only) an OMNIS list containing MIME body-parts. See the MailSplit command for a definition of the list. Note that you do not need to fill in the character set and content encoding type columns. SMTPSend will automatically use the ISO-8859-1 character set for text, the 7bit encoding for message content, quoted-printable encoding for text content, and base64 encoding for all other content types. If you wish to override this default behavior, you can.

CC specifies the carbon-copy recipients for the message. You pass this parameter in the same way as the *To* parameter.

BCC specifies the blind carbon-copy recipients for the message. You pass this parameter in the same way as the *To* parameter.

Name is an OMNIS Character field containing a personal name that will appear in the header to identify the user by a more descriptive name than just the e-mail address, for example, OMNIS Webmaster

StsProc is an optional parameter containing the name of an OMNIS method that SMTPSend calls with status messages as submission of the message to the SMTP server

proceeds. The method can display a status message to the user. SMTPSend calls the method with no parameters, and the status information in the variable #S1.

Pri is on OMNIS Short Integer field that sets the priority of the e-mail. It accepts a single value in the range of 1 through 5, a 1 (one) indicating the highest priority.

XtraHdrs is an optional parameter which enables you to specify some additional SMTP headers to be sent with the message. . It is a 2 column list. Column 1 is the header name excluding the colon, and column 2 is the header value. For example, you could place 'Disposition-Notification-To' in column 1, and an email address in column 2, to send a 'Disposition-Notification-To' header. Note that SMTPSend does not validate the header, or attempt to filter out illegal duplicates.

TCPAccept

Reversible: NO **Flag affected:** NO

Parameters: Socket

Returns: *Socket* (or an error code < 0)

Syntax: TCPAccept(*Socket*)

You use TCPAccept to accept an incoming connection request from another application.

The *Socket* parameter is a socket which is listening for incoming connections on a particular port. You must create this socket using TCPSocket, bind a port (and implicitly the local machine's IP address) to it using TCPBind, and start listening for connections by calling TCPListen, before you can call TCPAccept to accept incoming connections using the socket.

TCPAccept is affected by the blocking state of the *Socket* parameter. If the *Socket* parameter is blocking, TCPAccept waits until an incoming connection arrives, and then returns a new *Socket* for the connection to the remote application. If the *Socket* parameter is non-blocking, TCPAccept will return a new *Socket* if an incoming connection request is already queued on the listening socket; otherwise, it will return the error status –10035, which means that the call would block.

TCPAccept returns a long integer, which is either a new socket for the accepted connection, or an error code less than zero. The new socket has the same blocking mode as the listening socket. The listening socket continues to listen for further incoming connection requests.

TCPAddr2Name

Reversible: NO **Flag affected:** NO
Parameters: Address
Returns: *Hostname* (or an error code < 0)
Syntax: TCPAddr2Name(*Address*)

TCPAddr2Name is a domain name service command to resolve the hostname for a given IP address.

Address is an OMNIS Character field containing the IP address to convert to a hostname. The IP address is of the form 255.255.255.254

Hostname is an OMNIS Character field which receives a hostname which maps to the IP address. The hostname is of the form machine[*.domainame.dom*]

Note: This command fails if the address of a Domain Name Server has not been defined for your computer. Not all host IP Addresses may be known to the Domain Name Server. If the Domain Name Server is busy or unavailable, the command times out and returns an error. Defining often -used servers in a local host's file or using a caching Domain Name Server increases performance of this command.

TCPBind

Reversible: NO **Flag affected:** NO
Parameters: Socket, Service|Port
Returns: *Status*
Syntax: TCPBind(*Socket*,*Service*|*Port*)

TCPBind binds a socket created with TCPSocket to a particular local port.

Socket is an OMNIS Long Integer field, containing the number of the socket.

Service/Port is either an OMNIS integer field containing the number of the port to which the socket should be bound, or an OMNIS character field containing the name of a service which will be resolved to a port number by a local lookup.

Status is an OMNIS Long Integer field which receives the value zero for success, or an error code < 0 for failure.

TCPBlock

Reversible: NO **Flag affected:** NO

Parameters: Socket, Option
Returns: *Status*

Syntax: TCPBlock(*Socket,Option*)

The TCPBlock command makes a socket blocking or non-blocking.

The blocking state of a socket affects the commands TCPAccept, TCPReceive, TCPSend, and HTTPSend. If you use TCPBlock to change the blocking state of sockets returned for FTP connections, this could result in undesirable behavior of the FTP commands.

If a socket is blocking, the commands listed above wait until they can complete successfully; in other words, a receive waits until it has received some data, a send waits until it has sent some data, and an accept waits until an incoming connection request arrives.

If a socket is non-blocking, the commands listed above will complete successfully if they can do so immediately; if not, they will return the error code –10035, which means that the command needs to block before it can complete successfully.

Socket is an OMNIS Long Integer field containing a number identifying a valid socket.

Option is an OMNIS integer field. Non-zero means non-blocking and zero means blocking.

Status is an OMNIS Long Integer field which receives the value zero for success, or an error code < 0 for failure.

TCPClose

Reversible: NO **Flag affected:** NO

Parameters: Socket[,Option]
Returns: *Status*

Syntax: TCPClose(*Socket[,Option]*)

TCPClose closes, and depending on the *Option*, releases a *Socket*. When the socket is connected, this will result in the closure of the connection to the remote application. All new sockets returned by all Web commands, must eventually be released using TCPClose, to avoid resource leakage.

The most brutal form of TCPClose is an abortive close. In this case, no consideration is given to the state of the connection, or exchanges with the remote application, and the socket is closed and released immediately. This form of TCPClose is recommended for use in error handling situations.

The mildest form of TCPClose is a partial close. In this case, the socket is not released, and you will need to call TCPClose again to release the socket. A partial close initiates a disconnect of the TCP/IP connection, by sending a TCP/IP packet with the finish flag set.

This means that you can no longer send data to the remote application, but you can continue to receive data. The remote application will be informed of the partial close, when it receives zero bytes; in the case of the TCPReceive command, it will return a received character count of zero. At this point, the remote application can continue to send data, and when it has finished, it issues a complete close itself.

The remaining form of TCPClose is a complete close. In this form, TCPClose initiates a close of the connection if necessary, receives data on the connection until no more is available (to flush the connection), and releases the socket. This is recommended practice for TCP/IP connections.

What does this mean in practice? Consider two applications A1 and A2, communicating using TCP/IP. A1 can either do a partial close or a complete close. In both cases, A2 will receive zero bytes, indicating that disconnection has been initiated. A2 can continue to send, and when it has finished, it issues a complete close. A1 can receive the data sent by A2 provided that it only issued a partial close. Eventually A1 will receive zero bytes, at which point it issues a final complete close. At this point, the connection has been gracefully closed, and the sockets used by both A1 and A2 have been released.

Socket is an OMNIS Long Integer field containing a number representing a previously opened socket.

Option is an optional OMNIS Integer field, which has the value zero for a complete close, 1 for a partial close, and 2 for an abortive close. If omitted, it defaults to a complete close.

Status is an OMNIS Long Integer field which receives the value zero for success, or an error code < 0 for failure.

TCPConnect

Reversible: NO **Flag affected:** NO
Parameters: Hostname, Service|Port
Returns: *Socket* (or an error code < 0)
Syntax: TCPConnect(*Hostname, Service*|*Port*)

TCPConnect establishes a TCP/IP connection to a remote application, and returns a new socket representing that connection.

Hostname is an OMNIS Character field containing the hostname or IP address of the system on which the remote application is running.

Service/Port is either an OMNIS integer field containing the number of the port on which the remote application is listening for new connections, or an OMNIS character field containing the name of a service which will be resolved to a port number by a local lookup.

Socket is an OMNIS Long Integer field that receives either the number of the new socket, or an error code < 0.

Note: This differs from the more standard implementation of the sockets connect call. Instead of creating a socket with one command (such as TCPSocket), then passing the socket to a connect command, TCPConnect creates the socket and returns the socket number in one step.

TCPGetMyAddr

Reversible: NO **Flag affected:** NO
Parameters: [Socket]
Returns: *Address* (or an error code < 0)
Syntax: TCPGetMyAddr(*[Socket]*)

TCPGetMyAddr is a domain name service command to resolve the IP address of the local computer running OMNIS.

You can optionally pass a *Socket,* which corresponds to an open connection. In this case, the command returns the local IP address bound to the local endpoint of the connection. There are two cases where this is useful.

1. It is not a mandatory requirement that a WinSock API implementation can return the local IP address, without a socket for an open connection. In this case it is likely that TCPGetMyAddr will return 0.0.0.0.
2. If the local machine has more than one IP address, passing a socket eliminates ambiguity, and returns the local IP address used for the open connection.

Address is an OMNIS Character field which receives the IP Address of the local host. The IP address is of the form 255.255.255.254

TCPGetMyPort

Reversible: NO **Flag affected:** NO
Parameters: Socket
Returns: *Port* (or an error code < 0)
Syntax: TCPGetMyPort(*Socket*)

TCPGetMyPort is a command to return the number of the local TCP/IP port to which a given socket is bound.

Socket is an OMNIS Long Integer field containing a connected socket, or a socket bound to a port.

Port is an OMNIS Long Integer field which receives the port number, or an error code < 0.

TCPGetRemoteAddr

Reversible: NO **Flag affected:** NO

Parameters: Socket
Returns: *Address* (or an error code < 0)

Syntax: TCPGetRemoteAddr(*Socket*)

TCPGetRemoteAddr returns the IP address of the remote computer to which a given socket is connected.

Socket is an OMNIS Long Integer field containing a connected socket.

Address is an OMNIS Character field which receives the IP Address of the host to which the socket is connected. The IP address is of the form 255.255.255.254

TCPListen

Reversible: NO **Flag affected:** NO

Parameters: Socket
Returns: *Status*

Syntax: TCPListen(*Socket*)

TCPListen puts a socket created with TCPSocket into listening mode. When a socket is in listening mode, it will acknowledge incoming connection requests addressed to the port bound to the socket, and place them in a queue, ready to be accepted using TCPAccept.

Socket is an OMNIS Long Integer field containing the number of a socket that has been bound to a port.

Status is an OMNIS Long Integer field which receives the value zero for success, or an error code < 0 for failure.

TCPName2Addr

Reversible: NO **Flag affected:** NO

Parameters: Hostname
Returns: *Address* (or an error code < 0)

Syntax: TCPName2Addr(*Hostame*)

TCPName2Addr is a domain name service command that returns the IP address for a given *Hostname*.

Hostname is an OMNIS Character field containing a hostname to convert to an IP address. The hostname is of the form machine[.*domainame.dom*]

Address is an OMNIS Character field which receives the IP Address corresponding to the given hostname. The IP address is of the form 255.255.255.254

Note: This command fails if the address of a Domain Name Server has not been defined in your computer. Not all host IP Addresses may be known to the Domain Name Server. If the Domain Name Server is busy or unavailable, the command times out and returns an error. Defining often-used servers to a local host's file or using a caching Domain Name Server increases performance of this command.

TCPPing

Reversible: NO **Flag affected:** NO

Parameters: Hostname[,Size,Timeout]
Returns: *Milliseconds* (or an error code < 0)

Syntax: TCPPing(*Server*[,*Size*[,*Timeout*]])

TCPPing sends an ICMP request packet to a specified IP address or named host. It returns the round-trip packet time in milliseconds. If the host is unreachable or not available, the command will return a negative error code.

Hostname is an OMNIS Character field containing the IP address or hostname of the host to ping.

Size is an optional parameter. It is an OMNIS Long Integer field containing the size, in bytes, of the packet to ping the specified host. Typical values are from 512 to 2,048 bytes. The command makes sure the size is between 1 and 16k bytes, and will force sizes outside this range to the minimum or maximum, appropriately. If omitted, *Size* defaults to 256.

Timeout is an optional parameter. It is an OMNIS Long Integer field containing the number of milliseconds to use as a timeout value for the ping request. If the host is unavailable or does not respond in the specified number of milliseconds, the TCPPing function cancels the ping request and returns -1. If omitted, *Timeout* defaults to 3000.

Milliseconds is an OMNIS Long Integer field. When no error occurs, TCPPing returns the number of milliseconds that it took to receive the ping response from the host. On very fast

LANs, it is possible that the ping can complete so quickly that the value may be 0 (zero). A value of -1 (minus one) is returned if the ping times out. All other negative values are error codes.

TCPReceive

Reversible: NO **Flag affected:** NO

Parameters: Socket, Buffer

Returns: *ReceivedCharacterCount* (or an error code < 0)

Syntax: TCPReceive(*Socket,Buffer*)

TCPReceive receives data on a connected socket.

Socket is an OMNIS Long Integer field containing the socket number of a connected socket.

Buffer is an OMNIS Character or Binary field into which TCPReceive places the received data.

TCPReceive receives data into the buffer, and then returns the number of received characters to the OMNIS Long integer *ReceivedCharacterCount.* If an error occurs, TCPReceive returns a negative error code. Note that zero can be returned to *ReceivedCharacterCount* when graceful closure of the connection is initiated by the remote application, and there is no more data to receive. See TCPClose for details.

Note: Non-blocking sockets return an error code of -10035 if no data is available. Some implementations of socket libraries may have limits on the number of characters you can receive at one time. Consult the documentation for your installed sockets libraries. You may have to read data in multiple chunks to assemble an entire message. Always check the number of characters returned to make sure there was no error.

TCPSend

Reversible: NO **Flag affected:** NO

Parameters: Socket, Buffer

Returns: *SentCharacterCount* (or an error code < 0)

Syntax: TCPSend(*Socket, Buffer*)

TCPSend sends data on a connected socket.

Socket is an OMNIS Long Integer field containing the socket number of a connected socket.

Buffer is an OMNIS Character or Binary field containing the data to send on the socket.

TCPSend sends as much data as it can, and then returns the number of characters it sent to *SentCharacterCount*, an OMNIS Long Integer field. If an error occurs, TCPSend returns a negative error code

Note: Non-blocking sockets return an error code of -10035 if the socket cannot accept the characters to send immediately. Some implementations of socket libraries may have limits on the number of characters you can send at one time. Consult the documentation for your installed sockets libraries. You may have to send a message in multiple chunks in order to send a very long message. Always check the number of characters returned, to determine how much of the buffer has actually been sent; if the value is less than the buffer size, you need to call send again, to send the rest of the buffer.

TCPSocket

Reversible: NO **Flag affected:** NO

Parameters: None

Returns: *Socket* (or an error code < 0)

Syntax: TCPSocket()

TCPSocket creates a new socket. The only use of such a socket is to bind a port to it using TCPBind, start listening on the port using TCPListen, and then accept incoming connections using TCPAccept.

Socket is an OMNIS Long Integer field which receives the number of the allocated socket. If an error occurs, the command returns a negative number.

UUDecode

Reversible: NO **Flag affected:** NO

Parameters: Stream,DecodedStream
Returns: *Status*

Syntax: UUDecode(*Stream,DecodedStream*)

UUDecode turns Uuencoded information back into text or binary information. It is the inverse of UUEncode. Uuencoded information is commonly sent over the Internet in a manner that preserves binary information.

Stream is an OMNIS Character or Binary field containing the information to UUDecode.

DecodedStream is an OMNIS Character or Binary field that receives the resulting Uudecoded representation of the *Stream* argument. Because Uuencoding is generally used for binary information, a Binary field is the norm.

Status is an OMNIS Long Integer field which receives the value zero for success, or an error code < 0 for failure.

UUEncode

Reversible: NO **Flag affected:** NO

Parameters: Stream,EncodedStream
Returns: *Status*

Syntax: UUEncode(*Stream,EncodedStream*)

UUEncode turns a stream into an encoded stream of 64-character lines of print-only ASCII characters. The encoded version is approximately 1.25 times larger than the original.

Stream is an OMNIS Character or Binary field containing the information to UUEncode.

EncodedStream is an OMNIS Character or Binary field that receives the resulting Uuencoded representation of the S*tream* parameter.

Status is an OMNIS Long Integer field which receives the value zero for success, or an error code < 0 for failure.

WebDevSetConfig

Reversible: NO **Flag affected:** NO

Parameters: [ErrorProc,CommsTimeout]

Returns: *Status*

Syntax: WebDevSetConfig(*[Proc,CommsTimeout]*)

The WebDevSetConfig command allows you to set some configuration options for the WEB commands.

ErrorProc is the **WebDevError** method. WebDevError is an OMNIS method which ALL of the other WEB commands call when an error occurs. WEB command execution is as follows:

- Attempt to execute command
- If no error occurs, return successful status information.
- If an error occurs, and there is no WebDevError method, display an OK message to report the command. Return the error code. **Note:** if you are using the command on a server, it is undesirable for this OK message to be displayed ; in this case, set a WebDevError method.
- If an error occurs, and there is a WebDevError method, call the WebDevError method. Return the error code.

ErrorProc is an OMNIS Character field containing the name of the WebDevError method, for example MYLIBRARY.MYCODE/MYPROC.

When a WEB command calls the WebDevError method, it passes it three parameters:

1. A character parameter containing an error message.
2. A long integer containing the error code.
3. A character parameter containing the WEB command name.

To clear the WebDevError method, either pass no parameters, or an empty first parameter, to WebDevSetConfig.

You can also optionally pass *CommsTimeout* to this command. *CommsTimeout* is a long integer, which specifies the number of seconds that WEB commands will wait to receive data, before deciding that the remote application is not going to respond. **Note:** this time-out does not apply to TCPReceive. WebDevSetConfig multiplies this value by 60, to generate a value in $1/60^{th}$ second ticks, and stores the resulting unsigned long integer. If you pass an invalid value (such as zero), this will set the time-out to the default value of 60 seconds. If you do not pass a *CommsTimeout* parameter, the time-out remains unchanged.

The WebDevSetConfig command returns a long integer *Status.* Zero for success, or less than zero if an error occurs.

Web Command Error Codes

The WEB commands return negative error codes. Some of the error codes map directly on to the underlying API used for TCP/IP: namely, WinSock on the Windows platforms, Open Transport on the Power Macintosh, and MacTCP on the Macintosh 68k. Such error codes are platform specific. The remaining error codes are platform independent, and are generated directly by the WEB commands. We list these first, followed by separate listings of the WinSock, Open Tranport, and MacTCP error codes.

Platform Independent General Error Codes

These can apply to any command.

Error Code	Error Text
-501	Incorrect argument type
-502	Error getting information about an argument
-503	Incorrect number of parameters
-506	Unrecognised command
-507	Error locking handle
-508	Bad list generated by command
-509	Bad socket passed to command
-510	Type of socket passed to command is invalid (e.g. FTP socket passed to HTTP command)
-511	No address specified
-512	Could not open ICMP handle
-513	Could not start timer
-514	Service lookup failed (Mac only)
-515	Internet services provider not open (Mac only)
-516	The end-user cancelled the request
-517	Bad option passed to TCPClose
-518	No port bound to socket (Mac only)
-519	Socket is not listening - cannot accept (Mac only)
-521	Socket is not connected (Mac only)

-522	Timeout while waiting for response or request
-523	Badly formatted response from server
-524	Response from server is too short
-525	Response from server has incorrect syntax
-1010	Out of memory
-10035	The command would block

Platform Independent E-mail Command Error Codes

Error codes marked with * are received in responses from the server, and then returned as the result of command execution.

Error Code	Error Text
-1218	Parameter passed to mail command is too long
-1219	Unrecognised response from SMTP server
-1220*	SMTP: 211 System status, or system help reply
-1221*	SMTP: 214 Help message
-1222*	SMTP: 220 <domain> Service ready
-1223*	SMTP: 221 <domain> Service closing transmission channel
-1225*	SMTP: 251 User not local; will forward to <forward-path>
-1227*	SMTP: 421 <domain> Service not available, closing transmission channel
-1228*	SMTP: 450 Requested mail action not taken: mailbox unavailable [E.g., mailbox busy]
-1229*	SMTP: 451 Requested action aborted: local error in processing
-1230*	SMTP: 452 Requested action not taken: insufficient system storage
-1231*	SMTP: 500 Syntax error, command unrecognized
-1232*	SMTP: 501 Syntax error in parameters or arguments
-1233*	SMTP: 502 Command not implemented
-1234*	SMTP: 503 Bad sequence of commands
-1235*	SMTP: 504 Command parameter not implemented

-1236*	SMTP: 550 Requested action not taken: mailbox unavailable
-1237*	SMTP: 551 User not local; please try <forward-path>
-1238*	SMTP: 552 Requested mail action aborted: exceeded storage allocation
-1239*	SMTP: 553 Requested action not taken: mailbox name not allowed
-1240*	SMTP: 554 Transaction failed
-1241	Error decoding quoted printable or base 64 encoded MIME data
-1242	Body part list is inconsistent - cannot build MIME content
-1244	POP3: error received from server
-1245	POP3: could not extract message size from response to LIST command
-1246	POP3: message received from server is too large (does not match size in LIST command response)

Platform Independent FTP Command Error Codes

Error codes marked with * are received in responses from the FTP server, and then returned as the result of FTP command execution.

Error Code	**Error Text**
-1105*	Need FTP account for storing files
-1106*	Requested FTP action aborted: page type unknown
-1107*	Requested FTP file action aborted. Exceeded storage allocation (for current directory or dataset)
-1108*	Requested FTP action not taken. File name not allowed
-1109*	Requested FTP action aborted: local error in processing
-1110*	FTP file not found, or no access to file
-1116	Parameter passed to FTP command is too long
-1117	Parameter passed to FTP command contains invalid characters
-1119*	FTP Restart marker reply
-1120*	FTP serviceready in nnn minutes
-1121*	FTP data connection already open; transfer starting

-1122*	FTP file status okay; about to open data connection
-1123*	FTP user name okay, need password
-1124*	Unrecognised FTP positive preliminary reply
-1125*	Unrecognised FTP positive intermediate reply
-1126*	Unrecognised FTP transient negative completion reply
-1127*	Unrecognised FTP permanent negative completion reply
-1128	68k only: Server does not support passive FTP. Use FTPSetConfig to allocate ports for active FTP
-1129	Could not extract server IP address and port from response to FTP command PASV
-1130	FTP transfer type must be zero (for ASCII) or one (for binary)
-1131	FTP could not open local file
-1132	Error while FTP was reading or writing the local file
-1134*	Need account for FTP login
-1135*	Requested FTP file action pending further information
-1136*	FTP service not available, closing control connection
-1137*	Cannot open FTP data connection
-1138*	FTP connection closed; transfer aborted
-1139*	Requested FTP file action not taken. File unavailable (e.g., file busy)
-1142*	Requested FTP action not taken. Insufficient storage space in system
-1143*	Syntax error: FTP command unrecognized or too long
-1144*	Syntax error in FTP parameters or arguments
-1145*	FTP command not implemented
-1146*	Bad sequence of FTP commands
-1147*	FTP command not implemented for that parameter
-1148*	Not logged in to FTP server
-1149*	Unrecognised response from FTP server

Platform Independent HTTP Command Error Codes

Error Code	Error Text
-1154	Unable to determine end of HTTP header
-1161	Incomplete HTML tag
-1180	Parameter passed to HTTP command is too long
-1181	Post with CGI parameters sends CGI parameters as content: cannot supply content-type/length header in header list
-1182	Received HTTP request is badly formatted
-1183	Received HTTP request does not contain the HTTP version
-1184	Received HTTP request contains badly formatted CGI parameters
-1185	Invalid HTTP status code - must be 1-999
-1186	The client HTTP application closed the connection
-1187	The client HTTP application did not send a Content-Length header

WinSock Error Codes

Error Code	Error Text
-10004	Interrupted function call
-10009	Bad file descriptor
-10013	Permission denied
-10014	Bad address
-10022	Invalid argument
-10024	Too many open files
-10036	Operation now in progress
-10037	Operation already in progress
-10038	Socket operation on non-socket
-10039	Destination address required
-10040	Message too long
-10041	Protocol wrong type for socket

-10042	Bad protocol option
-10043	Protocol not supported
-10044	Socket type not supported
-10045	Operation not supported
-10046	Protocol family not supported
-10047	Address family not supported by protocol family
-10048	Address already in use
-10049	Cannot assign requested address
-10050	Network is down
-10051	Network is unreachable
-10052	Network dropped connection on reset
-10053	Software caused connection abort
-10054	Connection reset by peer
-10055	No buffer space available
-10056	Socket is already connected
-10057	Socket is not connected
-10058	Cannot send after socket shutdown
-10059	Too many references; cannot splice
-10060	Connection timed out
-10061	Connection refused
-10062	Too many levels of symbolic links
-10063	File name too long
-10064	Host is down
-10065	No route to host
-10066	Directory not empty
-10067	Too many processes
-10068	Too many users
-10069	Disk quota exceeded

-10070	Stale NFS file handle
-10071	Too many levels of remote in path
-10091	Network subsystem is unavailable
-10092	WINSOCK.DLL version out of range
-10093	Successful WSAStartup() not yet performed
-10101	Graceful shutdown in progress
-11001	Host not found
-11002	Non-authoritative host not found
-11003	This is a non-recoverable error
-11004	Valid name, no data record of requested type

Open Transport Error Codes

Error Code	**Error Text**
-2014	Bad address
-2016	Device or resource busy
-2039	Destination address required
-2042	Protocol not available
-2043	Protocol not supported
-2048	Address already in use
-2050	Network is down
-2051	Network is unreachable
-2052	Network dropped connection on reset
-2053	Software caused connection abort
-2054	Connection reset by peer
-2055	No buffer space available
-2056	Socket is already connected
-2057	Socket is not connected
-2059	Too many references: can't splice
-2060	Connection timed out

-2061	Connection refused
-2064	Host is down
-2065	No route to host
-3150	A bad address was specified
-3151	A bad option was specified
-3152	Missing access permission
-3153	Bad provider reference
-3154	No address was specified
-3155	Call issued in wrong state
-3156	Sequence specified does not exist
-3157	A system error occurred
-3158	An event occurred - call Look()
-3159	An illegal amount of data was specified
-3160	Passed buffer not big enough
-3161	Provider is flow-controlled
-3162	No data available for reading
-3163	No disconnect indication available
-3164	No Unit Data Error indication available
-3165	A Bad flag value was supplied
-3166	No orderly release indication available
-3167	Command is not supported
-3168	State is changing - try again later
-3169	Bad structure type requested for OTAlloc
-3170	A bad endpoint name was supplied
-3171	A Bind to an in-use addr with qlen > 0
-3172	Address requested is already in use
-3173	Accept failed because of pending listen
-3174	Tried to accept on incompatible endpoint

-3175	Endpoint must be bound with a qlen parameter equal to 0
-3176	The address to which this endpoint is bound differs from that of the endpoint that received the connection request
-3177	The maximum number of outstanding indications has been reached for the endpoint
-3178	An unspecified protocol error occurred
-3179	Attempted synchronous call at interrupt time
-3180	The command was cancelled
-3200	Permission denied
-3201	Endpoint mapper or entity not found
-3202	No such resource
-3203	Interrupted system service
-3204	I/O error
-3205	No such device or address
-3208	Bad file number
-3210	Try operation again later
-3211	Cannot allocate enough memory to meet your request
-3212	Permission denied
-3213	Bad address
-3215	Device or resource busy
-3216	Attempt to register a port or other entity that already exists
-3218	No such device
-3221	Invalid argument
-3224	Not a character device
-3231	Broken pipe
-3233	Message size too large for STREAM
-3234	Call would block so was aborted
-3236	kEALREADYErr
-3237	Socket operation on non-socket

-3238	Destination address required
-3239	Message too long
-3240	Protocol wrong type for socket
-3241	Protocol not available
-3242	Protocol not supported
-3243	Socket type not supported
-3244	Operation not supported on socket
-3247	Address already in use
-3248	Can't assign requested address
-3249	Network is down
-3250	Network is unreachable
-3251	Network dropped connection on reset
-3252	Software caused connection abort
-3253	Connection reset by peer
-3254	No buffer space available
-3255	Socket is already connected
-3256	Socket is not connected
-3257	Can't send after socket shutdown
-3258	Too many references: can't splice
-3259	Connection timed out
-3260	Connection refused
-3263	Host is down
-3264	No route to host
-3269	A catastrophic error has occurred which probably renders the underlying stream unusable
-3270	An Ioctlcommand has timed out instead of completing normally
-3271	Cannot allocate enough system resources to meet request
-3272	kEBADMSGErr
-3273	kECANCELErr

-3274	kENOSTRErr
-3275	kENODATAErr
-3276	kEINPROGRESSErr
-3277	kESRCHErr
-3278	kENOMSGErr
-3279	Client has not called InitOpenTransport or InitOpenTransportUtilities
-3280	The port your provider was using is disabled because it was unregistered
-3281	The port your provider was using is disabled because it was ejected
-3282	TCP/IP stack improperly configured by the TCP/IP control panel
-3283	Configuration changed
-3284	The port your provider was using is disabled because the user switched
-3285	The port your provider was using is disabled because it lost the connection

MacTCP Error Codes

Error Code	Error Text
-23000	Bad network configuration
-23001	Bad IP configuration error
-23002	Missing IP or LAP configuration error
-23003	Error in MacTCP load
-23004	Error in getting address
-23005	Connection is closing
-23006	Invalid length
-23007	Request conflicts with existing connection
-23008	Connection does not exist
-23009	Insufficient resources to perform request
-23010	Invalid stream pointer
-23011	Stream already open

-23012	Connection terminated
-23013	Invalid buffer pointer
-24014	Invalid RDS or WDS structure
-23015	Open failed
-23016	Command timeout
-23017	Duplicate socket
-23032	Packet too large to send without fragmenting
-23033	Destination not responding
-23035	ICMP echo timed-out
-23036	No memory to send fragmented packet
-23037	Cannot route packet off-net
-23041	Name syntax error
-23042	Cache fault
-23043	No result procedure
-23044	No name server
-23045	Name not found
-23046	No answer
-23047	The domain name server returned an error
-23048	Out of memory

Index